Finding Phishing Websites using Kernel Methods

Marc Fuentes Oncins
Carlos Hurtado Comín
Victor Novelle Moriano

20 of January of 2021

Contents

1 Objective

The objective of this project consists in developing specific kernel methods for categorical variables in order to apply them to a real world problem. As a result of that, this project consists in selecting and pre-processing the selected data, performing a previous study to acquire a deeper knowledge about them, developing the kernels and testing them in order to select the one that has the best performance. To do so, in the next pages we will study a database that includes data from 11055 web pages, analyzing them taking into consideration several factors such as URL Length, Redirects, PageRanks... among others to classify them into phishing or secure websites.

2 Motivation

We selected a *Phishing/Secure* websites data set because we consider that online security is a very interesting topic to analyze. Although this subject has been mentioned several times throughout the degree, we have never studied it in depth. If we focus on our specific theme, website security, it is being widely analyze nowadays, due to being a extremely complex problem located in an ever-evolving landscape. For these reasons, we decided to further gain knowledge about this topic and try to solve the malicious webs identification problem using kernel methods. Moreover, the selected data set is conformed only by categorical variables, fact that caught our attention positively, because it implied that we could generate kernels that took this into account instead of applying generic and already designed kernels functions.

3 Description of the data set

3.1 Data collection

The data set we will work with has been extracted from the data repository for machine learning *UC Irvine Machine Learning*[2]. The data set *PhishingWebsites* is original from a study carried out by three researches from two different universities, Rami Mustafa A Mohammad (University of Huddersfield), Lee McCluskey (University of Huddersfield), Fadi Thabtah (Canadian University of Dubai). In this data set, 30 factors were studied and are related to the risk that a page is malicious. As they state in the auxiliary document provided with the data set, there aren't many reliable training data sets to perform this classification task, due to the lack of consensus about which variables characterize phishing websites. To solve this, they analyzed 11055 different pages, established different thresholds for each of the factors taken into account in this analysis (which will be explained in the following section) to generate an accurate data set reliable enough to perform classification tasks.

3.2 Data set content

Once we have analyzed the origin and characteristics about the data collection process we can proceed to analyze the content of the set. As it has been previously mentioned, it has 11055 observations without any missing data.

Now, all the evaluated features will be explained, as well as the selected thresholds/decisions used by the authors, separating them into different sections:

3.2.1 Address Bar based Features

In this section we will consider all the metrics relating to the URL format, domains, structure among others.

1. **Using an IP Address**: Binary variable that indicates if an IP, in any format (*Hexadecimal e.g.*), is used in the domain.
   - -1 ≡ Has an IP (3793; 34.3%)
   - 1 ≡ Doesn't have an IP (7262; 65.7%)
2. **Long URL**: An usual ruse used by phishing websites consists in elaborating long URLs to hide doubtful parts in the non-showed section of the URL by the browser. To establish a threshold, the authors computed the average URL length of the data set (54), and classified the observations depending on whether they notably,lightly or didn't exceed this value.
   - -1 ≡ *length(URL)* > 75 (8960; 81.0%)
   - 0 ≡ 75 ≥*length(URL)* ≥ 54 (135; 1.2%)
   - 1 ≡ *length(URL)* < 54 (1960; 17.8%)
3. **TinyURL**: Checks if the website uses some type of URL shortener, to reduce the domain length.
   - -1 ≡ Doesn't use a shortener (1444; 13.1%)
   - 1 ≡ Uses a shortener (9611; 86.9%)
4. **Having "@"**: Checks if the website domain uses *"@"*.
   - -1 ≡ Uses "@" (1655; 15.0%)
   - 1 ≡ Doesn't use "@" (9400; 85.0%)
5. **Redirects using "//"**: The use of "//" means a redirection. In normal websites, this operator is located on the 6th or 7th position, depending if the URL start with *"HTTPS"* or *"HTTP"*.
   - -1 ≡ Uses "//" in a position > 7 (1429; 12.9%)
   - 1 ≡ Otherwise (9626; 87.1%)
6. **Prefix/Suffix using (-)**: Using (-) between prefixes and suffixes in the domain name is used by phishing websites to create a fake legitimate image.
   - -1 ≡ Uses "*(-)*" in the domain name (9590; 86.7%)
   - 1 ≡ Otherwise (1465; 13.3%)
7. **Sub domains**: This factor evaluates the number of sub domains contained in an URL. The number of points in the URL determine the number of sub domains. To evaluate the number of domains, the *www.* and the *ccTLD* were discarded by the authors.

- -1 ≡ *num_dots(URL)* > 2 (3363; 30.4%)
- 0 ≡ *num_dots(URL)* $= 2$ (3622; 32.8%)
- 1 ≡ *num_dots(URL)* $= 1$ (4070; 36.8%)

8. **HTTPS**: Evaluates if the webiste uses HTTPS, the issuer of the certificate is well-known and the certificate has an anticquity of, at least, a year.
   - 1 ≡ Uses HTTPS, Issuer is trusted and the age of the certificate ≥ 1 (6331; 57.3%)
   - 0 ≡ Uses HTTPS but Issuer isn't trusted (1167; 10.6%)
   - -1 ≡ Otherwise (3557; 32.1%)
9. **Time of domain registration**: Usually phishing websites have a short life whereas the trustworthy domains are paid for several years in advance, according to the data set authors' beliefs.
   - -1 ≡ Domain expires in ≤ 1 (7389; 66.8%)
   - 1 ≡ Otherwise (3666; 33.2%)
10. **Favicon**: A *Favicon* is an icon associated to a web page. If the *Favicon* of the website is loaded from a different domain than the used in the address bar we might be in front of a phishing attempt.
    - -1 ≡ *Favicon* loaded from an external domain ≤ 1 (2053; 18.6%)
    - 1 ≡ Otherwise (9002; 81.4%)
11. **Non-Standard Port**: Checks if the usual connection and protocol ports used bi the website are in their recommended for security status.
    - -1 ≡ Some port isn't on its Preferred status (1502; 13.6%)
    - 1 ≡ Otherwise (9553; 86.4%)
12. **HTTPS in the domain**: Checks if the token is used in the domain part instead of its usual location in order to trick the users (*http://https-www-website.com/*
    - -1 ≡ Contains HTPPS in the domain (1796; 16.2%)
    - 1 ≡ Otherwise (9259; 83.8%)

3.2.2 Abnormal based Features

In this section we will consider different metrics related to anchors, tags, links and other basic elements of the website.

1. **Request URL**: Checks if the external objects in the page are loaded from another domain or the embedded elements share the same domain as the webpage address.
   - 1 ≡ %*Request URL* $< 22\%$ (6560; 59.3%)
   - 0 ≡ $22\% \leq$ %*Request URL* $\leq 61\%$ (0; 0.0%)
   - -1 ≡ Otherwise (4495; 40.7%)

2. **URL of Anchor**: As well as the *Request URL*, evaluates if the elements defined using the anchor *"<a>"* have different domain names or links to other pages.
    - 1 ≡ $\%URL\ Anchor < 31\%$ (2436; 22.0%)
    - 0 ≡ $31\% \leq \%URL\ Anchor \leq 67\%$ (5337; 48.3%)
    - -1 ≡ Otherwise (3282; 29.7%)
3. **Links in <*Meta*>,<*Script*> *and* <*Link*> *tags***: This factor evaluates that the data contained in the mentioned tags, which we will refer to as M, are linked to the same domain of the webpage.

    PD: <Meta> stores metadata about the HTML,<Script> creates a client side script and <Link> retrieve other web resources.

    - 1 ≡ $\%Links\ in\ M < 17\%$ (2650; 24.0%)
    - 0 ≡ $17\% \leq \%Links\ in\ M \leq 81\%$ (4449; 40.2%)
    - -1 ≡ Otherwise (3956; 35.8%)
4. **Server from Handler (SFH)**: This factor evaluates how the webpage behaves when submitting information. It takes into account two factors: if the SFH doesn't perform an action and if its domain name is different from the one on the webpage.
    - -1 ≡ SFH empty or is *"about:blank"*, (doesn't perform an action) (8440; 76.3%)
    - 0 ≡ SFH refers to a different domain (761; 6.9%)
    - 1 ≡ Otherwise (1854; 16.8%)
5. **Submit Info to Email**: This factor evaluates if a webpage tries to redirect the personal information introduced by an user to a mail account instead of directing i to a server for processing.
    - -1 ≡ Using *"mail()"* or *"mailto:"* functions when submitting user's info (2014; 18.2%)
    - 1 ≡ Otherwise (1854; 81.8%)
6. **Abnormal URL**: Evaluates if the host of the domain's name is included in the URL, consulting the WHOIS database.
    - -1 ≡ The host name isn't in the URL (1629; 14.7%)
    - 1 ≡ Otherwise (9426; 85.3%)

3.2.3 HTML and JavaScript based features

In this section we will consider metrics related to the internal code structure of the webpage. That consits evaluating the HTML and JavaScript code.

1. **Website forwarding**: Takes into account how many redirections have been applied to the website.
    - 1 ≡ $\#Redirections \leq 1$ (1279; 11.6%)

- 0 ≡ $2 \leq \#Redirections < 4$ (9776; 88.4%)
- -1 ≡ Otherwise (0; 0.0%)

2. **Status bar customization**: Takes into account if the webpage shows a fake URL to the user when hovering with the mouse on the address bar. This is usually implemented using *onMouseOver* on the source code.
    - -1 ≡ *onMouseOver* changes the status bar (1315; 11.9%)
    - 1 ≡ Otherwise (9740; 88.1%)
3. **Disabling right click**: Phishers disable the right click in order to avoid the users viewing and inspecting the webpage source code. This is usually implemented disabling the *event.button==2* on the source code.
    - -1 ≡ Right click disabled (476; 4.3%)
    - 1 ≡ Otherwise (10579; 95.7%)
4. **Using Pop-up window**: Checks if the website contains Pop-up windows that allow the users to introduce information (they contain text fields).
    - -1 ≡ Pop-up window with text fields (2137; 19.3%)
    - 1 ≡ Otherwise (8918; 80.7%)
5. ***IFrame* redirection**: *IFrame* is an HTML tag that allows displaying an additional webpage into the one that is shown. This could be used to display a malicious website in top of another, removing the borders of the phishing one.
    - -1 ≡ Uses *IFrame* (1012; 9.2%)
    - 1 ≡ Otherwise (10043; 90.8%)

3.2.4 Domain based features

In this section we will consider all the metrics relating to the domain features, taking into account it's age and recognition among others.

1. **Age of the domain**: Feature extracted from the WHOIS database. Generally, phishing websites have a short life span.
    - 1 ≡ Age of the domain ≥ 6 months (5866; 53.1%)
    - -1 ≡ Otherwise (5189; 46.9%)
2. **DNS record**: Checks if a DNS record is associated to the website domain or if it's empty or not found.
    - -1 ≡ No DNS record for the domain (3443; 31.1%)
    - 1 ≡ Otherwise (7612; 68.9%)
3. **Website traffic**: Evaluates the popularity of the website taking into account the number of visitors and the number of pages they visit. To do so, the *Alexa* database is consulted.
    - 1 ≡ Website rank < 100000 (5831; 52.7%)
    - 0 ≡ Website rank > 100000 (2569; 23.2%)

- -1 ≡ The webpage is not recognized in the database (2655; 24.1%)

4. **PageRank**: Consults the PageRank of the observations. This metric measures how important is a website on the Internet.
   - -1 ≡ PageRank < 0.2 (2854; 25.8%)
   - 1 ≡ Otherwise (8201; 74.2%)
5. **Google Index**: Examines whether a website is in Google's index or not. Usually, as phishing websites have a short life span, they may not be found on the *GI*.
   - 1 ≡ Website indexed by Google (9516; 86.1%)
   - -1 ≡ Otherwise (1539; 13.9%)
6. **Number of links pointing to the page**: Takes into account the number of external links pointing to the website. It's a measure of the legitimacy level of the site.
   - -1 ≡ # links pointing to the page = 0 (548; 4.9%)
   - 0 ≡ $0 <$ # links pointing to the page ≤ 2 (6156; 55.7%)
   - 1 ≡ Otherwise (4351; 39.4%)
7. **Statistical-reports based feature**: Takes into account statistical reports done by *PhishTank* and *StopBadware* about phishing websites. In this, it's checked if the website appears in the "Top 10 Malicious Domains" and "Top 10 Malicious IPs" rankings on these reports.
   - -1 ≡ Host belongs in the top rankings (1550; 14.0%)
   - 1 ≡ Otherwise (9505; 86.0%)

Finally, we have the target variable, which indicates if a website is phishing or not, and that will be used as true label to perform the classification task.

Result

- -1 ≡ Phishing (4898; 44.3%)
- 1 ≡ Otherwise (6157; 55.7%)

Thus,to develop our kernels we have a total of 30 categorical variables with, at most, 3 levels and the binarized response variable that indicates whether a website is phishing or not.

4 Exploratory analysis

Before training any model for the prediction of the nature of each webpage we must perform an exploratory analysis of the data to understand what we are working with and to ease the future performance of the models.

First of all we want to see how the data distribution is for each variable. Knowing how the data is distributed for the different features can improve future analysis and give clues about what we must focus on when using this data source. To do so we will show an histogram for each variable. For continuous variables the histogram can give really valuable information. In our case all the variables are binary or ternary and therefore we can't expect to extract the same amount of insight as if we were in the first scenario.

We now have 30 different histograms where many of them show a close to 50/50 distribution. We can't visually extract any value but a trained model will be able to use this data to classify web pages. Among those 30 histograms there are a few which differ from the rest. These are: Prefix and suffix (6), Subdomain (7), SSL (8), URL of anchor (14) and Web traffic (26).

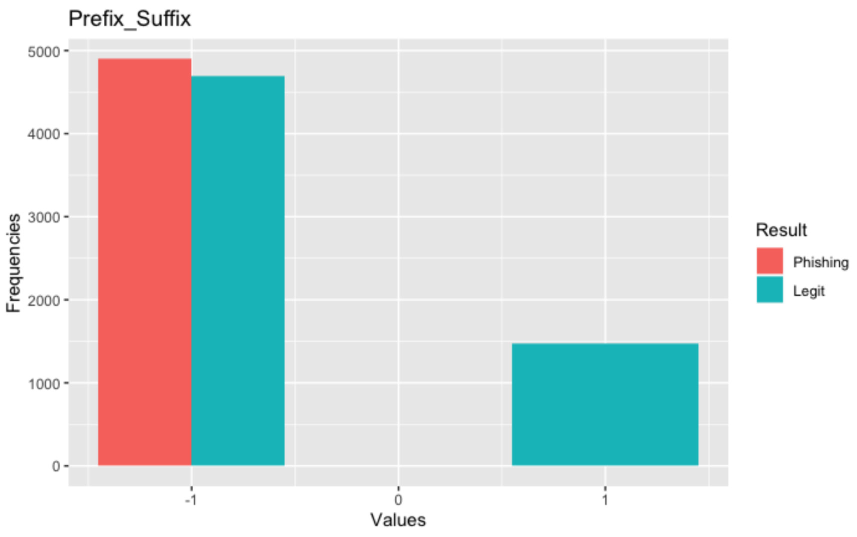

Figure 1: Distribution for Prefix_Suffix variable

In this case the variable encodes whether the url has a prefix or suffix separated by (-) in it. In the histogram we can see that for the webs that do present this the distribution is equal for both classes while the pages that don't use prefixes or suffixes are all legitimate web pages. This gives very solid information about the nature of the data. With the data we have and by looking at fig. 1 we can assume that any given url with no prefix or suffix should be legitimate.

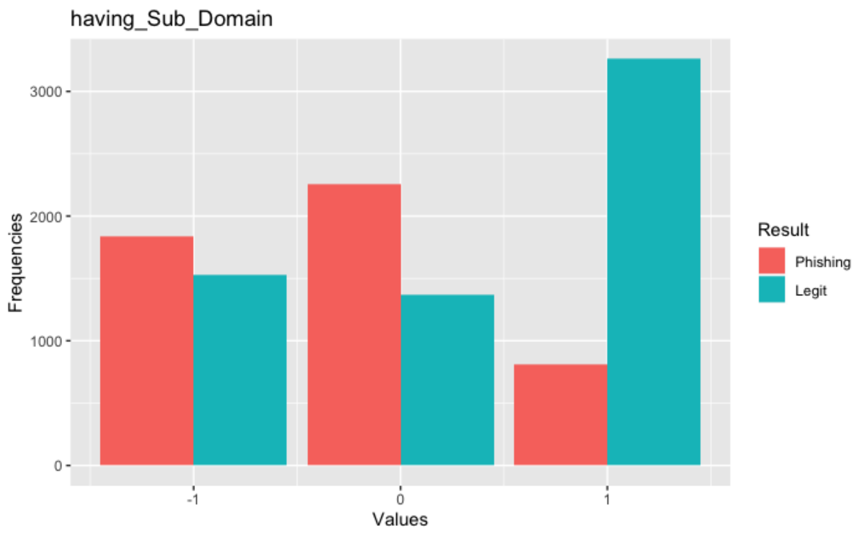

Figure 2: Distribution for having_Sub_Domain variable

This variable encodes whether a web page has 0, 1 or more subdomains. We can see how for the phishing pages the amount of them having one or more subdomains (-1, 0) is higher than the ones without. For the legit pages we find the opposite behaviour, the webs are concentrated in the no subdomain group with a smaller amount of 1 and more.

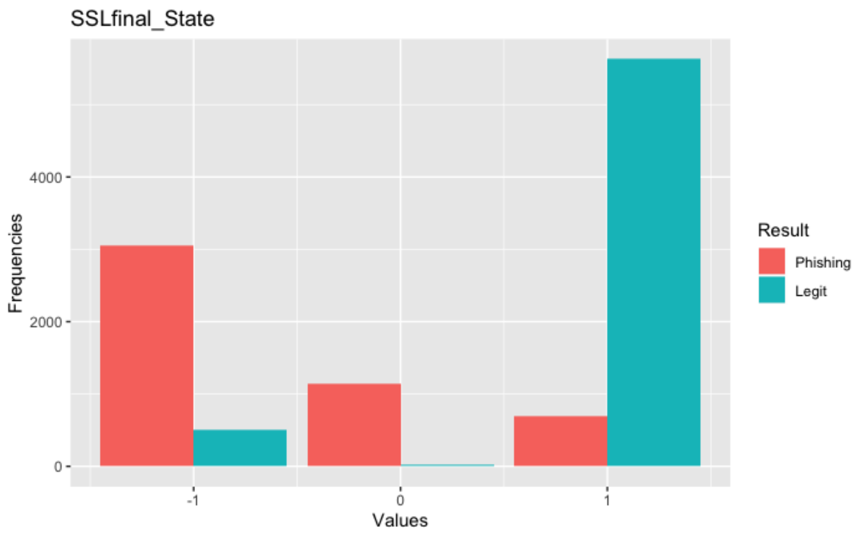

Figure 3: Distribution for SSLfinal_State

This variable encodes whether a web presents the HTTPS existence with an antiquity of more than 2 years. We can see how most of the legit pages have had HTTPS for more than 2 years while the phishing pages tend to have recent HTTPS or no at all.

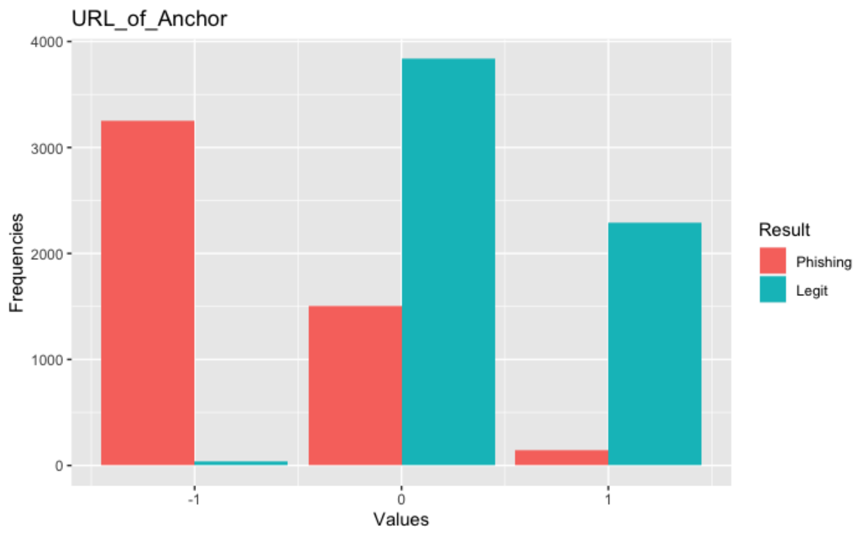

Figure 4: Distribution for URL_of_Anchor

This variable encodes whether an element of the web is constructed suspiciously or not. This element, the anchor, allows us to easily differentiate the legit and phishing in both extremes while for the middle value the distribution is not as abrupt. We can see how phishing webs are concentrated in the (-1, 0) groups while the legit ones are in (0,1). The problem with this variable is the indecision in the group 0, but as we will be working with ML models, every variable will contribute to the final result and complement each other so this won't be a problem.

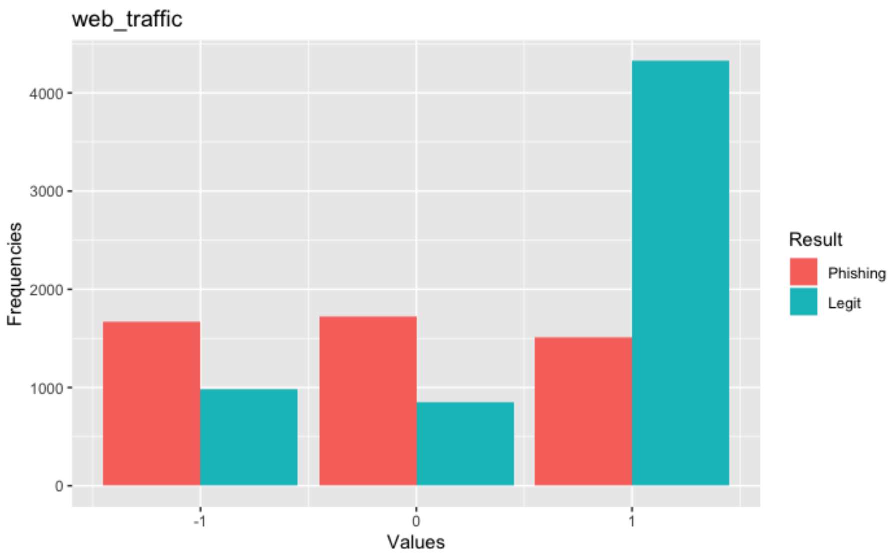

Figure 5: Distribution for web_traffic

This variable encodes the ranking of a page based on the Alexa dataset. We can see how the distribution for phishing pages is almost uniform but the legit ones tend to the group 1 which corresponds to pages in the top 100.000.

Then we will use Multi-Correspondence Analysis (MCA) to see how different variables correlate with the page class. The variables chosen for the study are biased towards finding phishing pages and we can expect some of them to give important discriminant information about the pages.

After performing MCA on the target variable with the others we could not extract much information. In our case using all variables for the analysis is not suitable as they clutter the image and no value can be extracted from it.

Taking into account the previous variable distribution study and the need to reduce the MCA variables, we will study the MCA for the variables that present differentiable distributions.

First we will use the features Prefix suffix, Anchor URL and Web traffic. As we have seen earlier these variables are good to detect the legit webs.

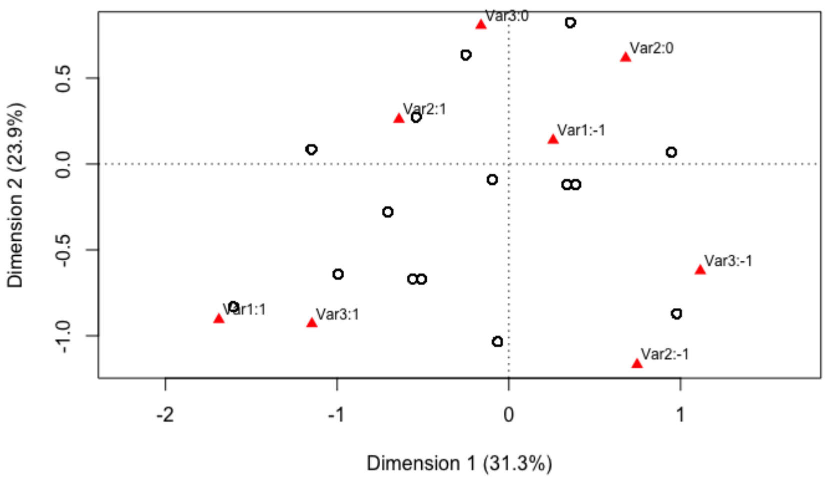

Figure 6: MCA using Prefix suffix, Anchor URL and Web traffic

We can see how the data is split into three groups. The leftmost group corresponds to pages with high pagerank that don't use prefix/suffix, these are likely legit pages. The one in the middle corresponds to pages that have low pagerank, and use prefix/suffix and have Anchor URL legit or suspicious; this is an intermediate group for pages that are suspicious. At the right we can find the last group which corresponds to malicious usage of Anchor URL and pages not registered in the pagerank. These ones are more likely phishing pages.

The first dimension of this MCA splits between legit and phishing with the intermediate group of indecision while the second dimension separates clearly legit/phishing pages from indecise suspicious ones.

We can now readjust some of the variables. Instead of using Prefix suffix we will use SSL final state (HTTPS) which is ternary just like the other two. By doing this we get the following result:

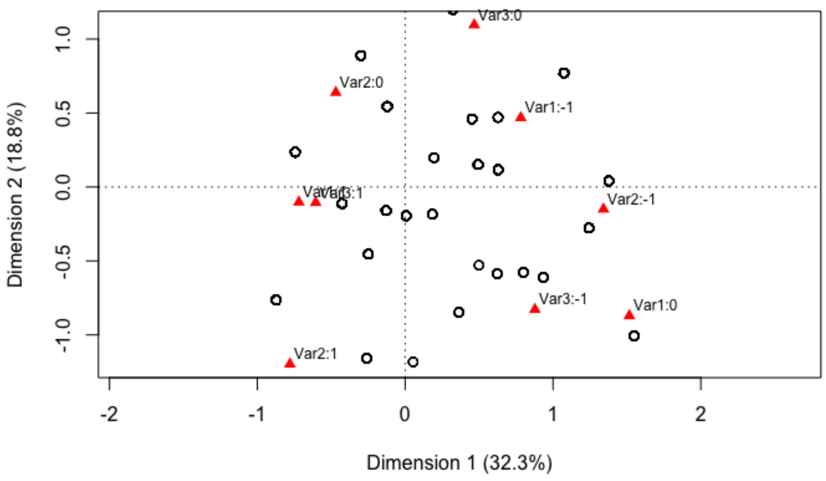

Figure 7: MCA plot for SSL final state, Anchor URL and Web traffic.

In this analysis we can't see any distant groups like in the first one but it seems like the plot is split in x=0. There is a group of parameters at the left and another at the right which seem like being related. If we analyze what each means we can see that the left group corresponds to legit pages with high pagerank, HTTPS existence and Anchor URL correct behaviour. The one at the right corresponds to suspicious and phishing pages, where variables are in the value 0 (suspicious) or they are at -1 (phishing). This group has pages with no pagerank (the lower pagerank, Var3:0 is farther from the group) and with recent or no HTTPS. This indicates those are probably not legit pages.

Here the first dimension splits the data between legit and not-so-legit with an intermediate step being low pagerank (Var3:0). The second one seems to split the pages between clearly legit or phishing and suspicious, as happened earlier, although it is now less differentiable.

Lastly we decided to explore the data from the kernel point of view. Therefore we used kernelized Principal Component Analysis (kPCA). PCA describes the data based on new non-correlated components. This technique can be kernelized and exploit the benefits of kernelization. The final result is our dataset expressed in terms of two new components derived from the data itself and with the potential of kernelization. Ideally we want our kPCA to allow us to linearly separate our data.

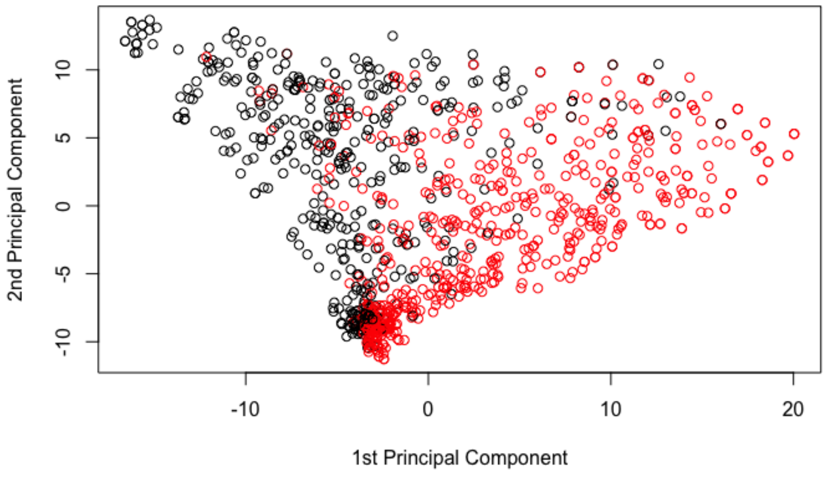

Figure 8: 2 principal components of kPCA

The result we obtain does a great job differentiating both groups (red and black). Although it is not linsep we can see how both clouds are more concentrated in the first component's extremes. The second component doesn't give any visual information.

5 Training

5.1 Splitting data in training and test

In order to study the different models that we will apply to the data, this data has been split into two sets. On the one hand, the first set includes 66.67% of the observations and will be used to choose the best model using cross validation. This set is called *training set*. On the other hand, the second partition has the other 33.33% of the data, and will be used to evaluate the selected models. This other set is called the *test set*, and allows us to see the capacity of the models to generalize, observing their behaviour on data they have never seen. This way, the training set will be used to find the proposed models that fit best our data base.

5.2 Models chosen for training

Given the nature of our data, which consists only on categorical explanatory variables, we might be led to think that the possible predictors we can use is very limited, but this is far from the truth. In this project we will focus on the application of kernel methods built around categorical variables. Then the observations are classified using Support Vector Machines (SVMs) in the feature space. The main issue at hand when using kernel methods is choosing an appropriate kernel function for the data. The majority of kernels work with quantitative data (i.e. vectors in R^d). We approach the task of finding an adequate function in different ways, from the most basic treatment of the data to the design of complex kernel functions that capture the similarities between the tuples.[1][3]

We use the library *kernlab*, which is very flexible since it allows us to use our own custom kernels with the library's methods. We have also used traditional classification algorithms such as Random Forests and Naive-Bayes so as to compare their performance with the kernel methods we designed. These kernel methods include the overlap kernel, different multivariate kernels, the RBF kernel and finally some kernels based on the similarity measures proposed by Gower and Legendre, all which will be explained in full detail in the following lines.

Most of these models require parameters for the training phase which modify their prediction behaviour and capacity. To choose the optimal hyper-parameters for each model we apply *5-times 5-fold Cross Validation* over the training data. This method consists on computing the average error of 25 instances of the model for each of the different values we consider for the given parameter. This way, from a random sample of the training data we train and validate the model with a partition of 80% to 20% respectively. This process is repeated 5 times and in the end we choose the value that results in the smallest error.

5.2.1 Random Forests

With *Random Forests* we have to adjust the number of trees that will make the *ensemble*. A large number of trees does not cause *over fitting*, instead as we increase the number of trees we approach a local minimum, that we achieve as the number of trees tends to infinity. Despite this, convergence is not linear, but instead the error decreases exponentially as a function of the number of trees and when it reaches a given size the improvement obtained from increasing the complexity of the forest is not greatly significant. This way, the behaviour we observed as we increase the size of the model trained using the library *randomForest* is the following:

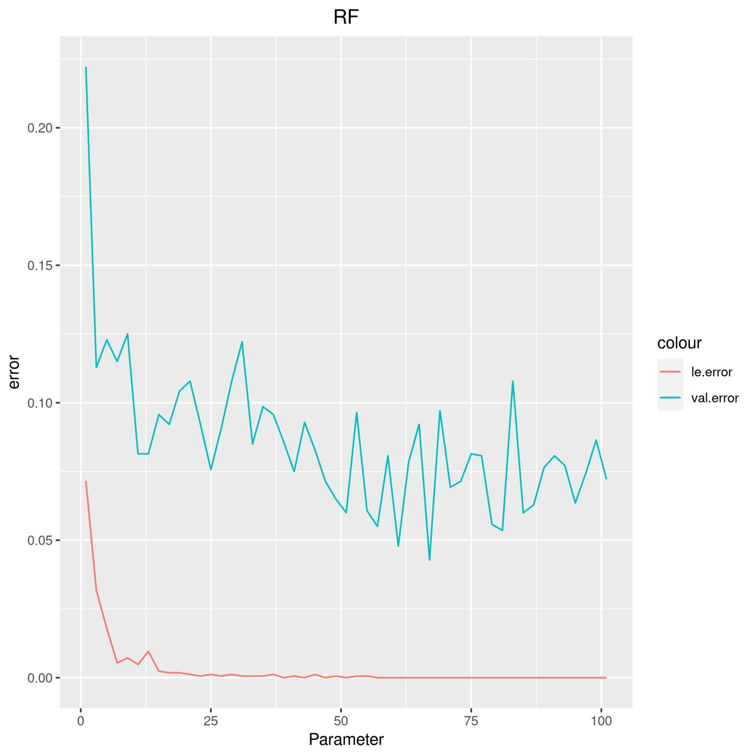

Figure 9: Error as a function of the number of trees

It can be observed that once we reach a certain size of the forest the error stabilizes and the increase in performance as a function of the number of trees is not significant. This is why we choose a number of trees large enough to obtain a small error but not so big as to make the computational cost unnecessarily large without improving performance. In this case, we can observe that at around 60 trees the error is stable and increasing the size does provoke a significant decrease in the prediction error.

5.2.2 Naive Bayes

Another clear choice for a model is Naive Bayes, since all of our variables are of a categorical nature. We have used the library *naivebayes* in order to use this classifier. It is important to point out that we have not used Laplacian correction, since the training set is very large and the variables have at most 3 different classes, so the chance of finding an unknown class during testing is almost 0. This model has no parameters that we have to fine tune and can be applied directly on the data.

5.2.3 The Overlap Kernel

The most intuitive and simple kernel function is the Overlap kernel, where the similarity is proportional to the number of variables on which 2 observations match. It can be expressed as:

$$k_O\left(x_i, x_j\right) = \frac{1}{d}\sum_{i=1}^{d} \mathbb{1}_{\left\{x_{ik}=x_{jk}\right\}}$$

Adapting binary matches to categorical variables it becomes necessary to define what is to be a match and what is not. As it has been explained previously, our data defines certain situations that might indicate a site is phishing, suspicious or legitimate. It is not straightforward whether we should consider all matches as equal. Thus, in addition to the Overlap Kernel we have implemented a Kernel based on the Simple Matching Coefficient (SMC), where we consider a match only those variables such that both are either suspicious or phishing.

5.2.4 The RBF Kernel

Another popular kernel function is the Gaussian Radial Basis Function (RBF). It relies on the Euclidean distance between observations, but since there is no Euclidean distance defined between categorical variables we have re-coded the data into numerical values. This way, values encoded as *phishing* will be considered -1, *suspicious* a 0 and *legitimate* sites a 1. Then *phishing* and *legitimate* sites will be at a distance of 2 while at the same time both will be at a distance of 1 from the *suspicious* sites and, as should be, a distance 0 from themselves.

We have used the built-in RBF kernel function from the *kernlab* library. Its mathematical formulation is the following:

$$k_{RBF}\left(x_i, x_j\right) = \exp\left(-\gamma \left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2\right)$$

The parameter γ can be fine-tuned, which can vary the performance of the model. This is shown further in the Annex.

5.2.5 Multivariate Kernels

These kernels are designed specifically for categorical variables. As opposed to the Overlap Kernel, we want the similarity of the pairs compared to represent their probability of occurrence. This way, we want their similarity to be higher when they coincide and they are less probable. As in the Overlap kernel, when they do not coincide the similarity will still be zero. In short, these kernels take into account the probabilistic structure of the data.

Accordingly, the univariate kernel the following, where Z is a categorical variable and P_Z its Probability Mass Function. This value is passed through an inverting function h_α, which makes small values of P_Z larger and big values smaller.

$$k^U\left(z_i, z_j\right) = \begin{cases} h_\alpha\left(P_Z\left(z_i\right)\right)\ if\ z_i = z_j \\ 0\ if\ z_i \neq z_j \end{cases}$$

Where h_α is defined as follows:

$$h_\alpha(z) = \left(1 - z^\alpha\right)^{1/\alpha}, \quad \alpha > 0$$

The previous kernel will return a value for a single dimension. We can generalize this kernel into a multivariate kernel by aggregating over d univariate comparisons. Moreover, we can expand the family of kernels by applying functions and over the similarities. These must preserve positive semi-definiteness. We have defined a multivariate kernel where we apply the exponential function over the sum scaled by a parameter γ, the bare bone multivariate kernel or identity multivariate kernel, and finally a centered multivariate kernel, similar to the first but we subtract the probability of of each value separately, passed through the inverting function.

$$k^{MV}(\mathbf{x},\mathbf{y}) = \exp\left(\frac{\gamma}{d}\sum_{i=1}^{d} k^{U}(x_i,y_i)\right)$$

$$k_{ID}^{MV}(\mathbf{x},\mathbf{y}) = \frac{1}{d}\sum_{i=1}^{d} k^{U}(x_i,y_i)$$

$$k_{C}^{MV}(\mathbf{x},\mathbf{y}) = \exp\left(\frac{\gamma}{d}\sum_{i=1}^{d}\left[2k^{U}(x_i,y_i) - k_1^{U}(x_i,x_i) - k^{U}(y_i,y_i)\right]\right)$$

These kernels depend on the hyper-parameters α, from the inverting function, and γ, the value we multiply by in the exponential function. The variations in the performance for the different values of the parameters are shown with great detail in the Annex.

5.2.6 Kernels based on Gower Legendre similarities

Gower and Legendre define a series of similarity measures in their work *Metric and Euclidean properties of dissimilarity coefficients*, some of which are positive semi-definite. Thus, these similarities have been taken and adapted into kernel functions that work for categorical variables. We have expanded the previous work on using these on binary variables and adapted them for variables with 3 possible values.

Most of the variables do not denote absence or presence but instead the different situations that can arise. First we need to redefine what is a match. Phishing - Phishing would be the binary match but we need to scale this for the ternary case. In this study, the combination Suspicious - Suspicious can be also included in the match category rather than in the irrelevance due to the possibility of both of them being phishing.

For the dissimilarity case (b and c) we will have all the pairs where the values differ. To correctly define b and c we need to divide all the possible different pairs into two groups of equal size.

Lastly, for the irrelevant case we decided to stick to the binary case, where only Legit -Legit is considered. Then the possible pairs are the following:

$$\begin{aligned}
&a \rightarrow \{-1,-1\}\{0,0\}\\
&b \rightarrow \{1,0\}\{1,-1\}\{0,-1\}\\
&c \rightarrow \{-1,0\}\{-1,1\}\{0,1\}\\
&d \rightarrow \{1,1\}
\end{aligned}$$

The resulting sets are complete and disjoint, and we can define the rules by computing the frequency of each pair.

Alternatively, we can make use of the probabilistic nature of the data and define rules that depend on the previously defined probability information. We give the option to use either in the code. :

$$f(x) = h_{\alpha}(P_X(x))$$

These rules are defined as:

$$\begin{aligned}
a &= \sum_{x_i = y_i} [f(x_i) + f(y_i)] \\
b &= \sum_{x_i \neq y_i} f(x_i) \\
c &= \sum_{x_i \neq y_i} f(y_i) \\
d &= \sum_{x_i \neq y_i} [1 - f(x_i) + 1 - f(y_i)]
\end{aligned}$$

During testing we saw very similar results using both sets of rules. For the later tests we decided to use the probabilistic approach since it is computationally more efficient.

Every similarity measure is defined as a combination of this rules, and they can be computed by first obtaining the values of a, b, c and d and then substituting. The positive semi-definite similarity measures defined by Gower and Legendre are:

$$\begin{aligned}
GL3 &= \frac{a}{a+b+c} \\
GL4 &= \frac{a+d}{a+b+c+d} \\
GL5 &= \frac{a}{a+2(b+c)} \\
GL6 &= \frac{a+d}{a+2(b+c)+d} \\
GL7 &= \frac{a}{a+\frac{1}{2}(b+c)} \\
GL9 &= \frac{a-b-c+d}{a+b+c+d} \\
GL12 &= \frac{a}{\sqrt{(a+b)(a+c)}} \\
GL13 &= \frac{a}{\sqrt{(a+b)(a+c)(d+b)(d+c)}} \\
GL14 &= \frac{ad-bc}{\sqrt{(a+b)(a+c)(d+b)(d+c)}}
\end{aligned}$$

Finally, as with the multivariate kernels, we can fine-tune the the parameter α from the inverting function. In the annex we show plots with the differences in performance of each model when changing the parameter α.

5.3 Model validation

Once we have proposed some models for training, we have to decide which are useful to model our data. Furthermore, in a similar manner as when we fine-tuned the hyper-parameters, the best models have been chosen using *5-times 5-fold cross validation*, where we adjust each model with 80% of the training data and we validate it with the remaining 20%. This process is repeated 25 times (5 permutations of the database and 5 folds for each permutation). This allows us to obtain a robust estimation for the error of each model and choose the one that better fits the data. The results obtained for the validation errors for each model are the following:

RF	*Naive Bayes*	*Overlap*	*SMC*	*Multivariate*	*Multivariate Id*	*Multivariate C*
13.27	36.61	7.42	12.13	3.37	8.76	4.05

RBF	*GL3*	*GL4*	*GL5*	*GL6*	*GL7*	*GL9*	*GL12*	*GL13*	*GL14*
0.0	8.37	9.58	4.46	4.32	10.53	5.67	10.67	3.24	5.26

Table 1: 5x5 validation error

In order to provide a more graphical visualization of the errors and to put forward more information than just the average, we will present them in form of boxplots. This were obtained storing the error value for each one of the 25 iterations performed.

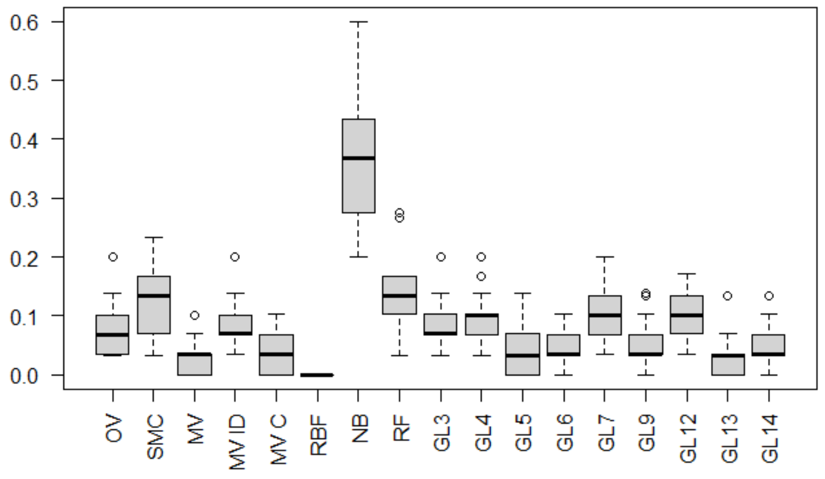

Figure 10: Boxplot of the errors

As it can be seen, the Gower Legendre based methods have an overall nice performance, as well as the Multivariate and RBF kernels. Those kernels achieve an accuracy notably higher than the rest, where we can observe different behaviours. In general, the designed kernels perform correctly, and in no case exceeded the 15 % error boundary. On the other hand, the already pre-defined R methods, have a poorer performance. This may be due to the fact that they not take into account the nature of the data and they do not allow us to exploit the inherent relationships present in it.

In order to obtain a more detailed view of the errors, we have selected those than in the previous boxplot have smaller values, to modify the y-axis scale and select the best ones.

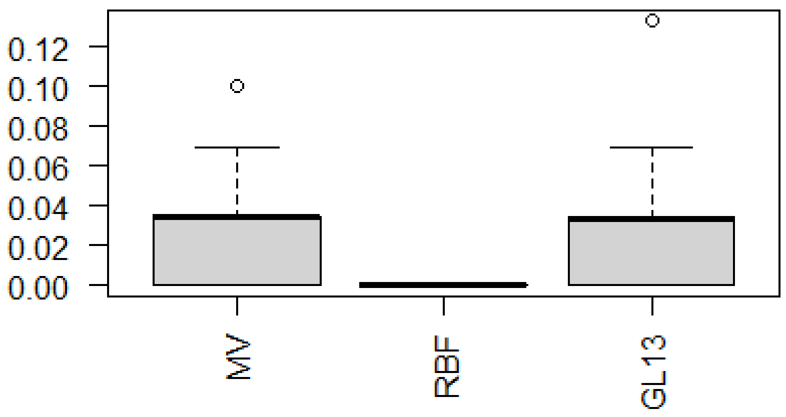

Figure 11: Boxplot of the best kernels/methods

As it can be seen, RBF achieve 0 validation error, whereas the other two have a very similar boxplot. While looking at the table presented above, we can see that GL13 have a smaller error despite having a same structured boxplot.

6 Test

After applying cross validation the best models turned out to be RBF, GL13 and MV with respective errors of (0.0%, 3.24%, 3.37%). Now we will train these models with the full train dataset and then proceed to evaluate their performance on the test one. The obtained results are:

RBF	*GL13*	*MV*
15.61	7.40	5.75

As it can be seen, the RBF kernel is the one with the highest test error despite being the one with the best performance over the training data. This might imply that there is some degree of over-fitting. In order to avoid that, we could perform several tests varying the *gamma* parameter or using a subset of the features. In relation to the two designed methods, they perform quite similarly, being the best the multivariate kernel. In general, we can conclude that the classification task is performed nicely, specially with the ”Multivariate” kernel, as less than 6 % of error has been achieved, demonstrating that the design of kernels specific to solve a problem represents a powerful alternative to the use of generic pre-established methods.

7 Conclusions

With this project, we can confirm that kernel methods represent a strong technique to achieve good results in classification tasks. Although we previously knew the potential that kernelization has on continuous variables, we liked to explore if this was true if we're located in a categorical environment. We have found out that despite not all kernels are suitable for this task, some generic ones can be used and new specific methods specifically designed for this problem can be created. Overall, we have obtained good results for the designed kernels and already existing ones. In future problems, where categorical variables are present, we could use some of the ones presented on this project to fulfill the required task successfully.

For this specific task we were able to achieve an error rate of 6%. Because of this, we consider that the selected kernels are able to accomplish the task it was meant to. After performing the exploratory analysis, we didn't expect achieving the accuracy reached but after the usage of kernelized methods we were able to achieve those rates. This is due to the fact that kernel methods work in high dimensionality spaces, easing the separation task.

Overall, this project allowed us to acquire a deeper knowledge about kernels' functionality and their potential as well as further explore a topic we are very interested in. We believe that kernel methods have far more to offer than what we might think at first glance, and this project has motivated us to continue exploring on this topic.

8 Annex

8.1 Hyper parameter optimization

8.1.1 Alpha

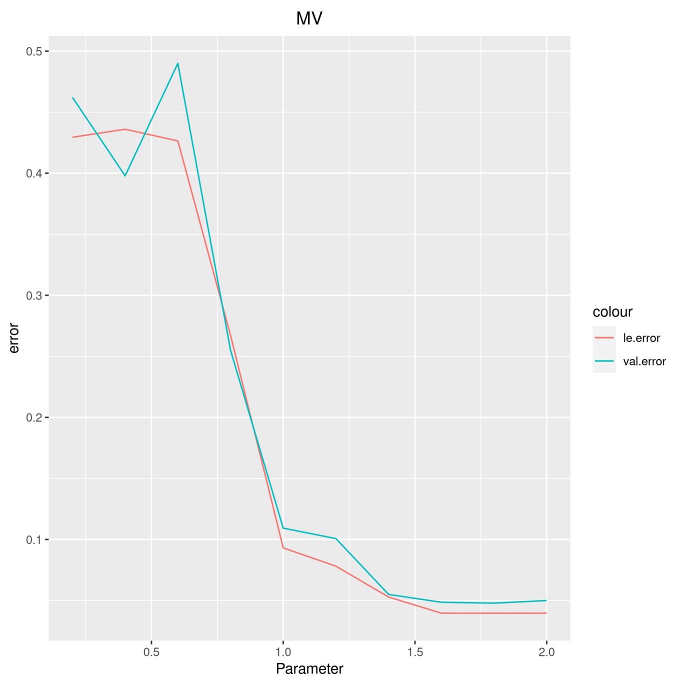

Figure 12: Error as a function of α

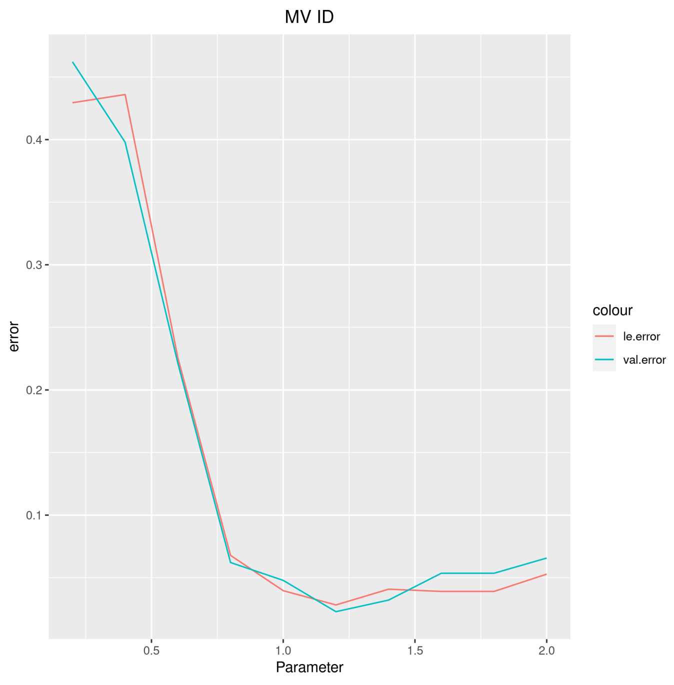

Figure 13: Error as a function of α

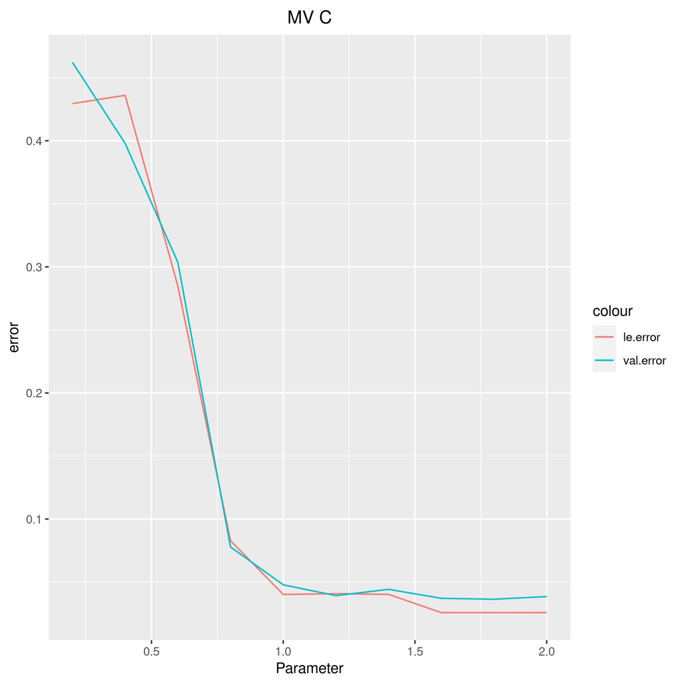

Figure 14: Error as a function of α

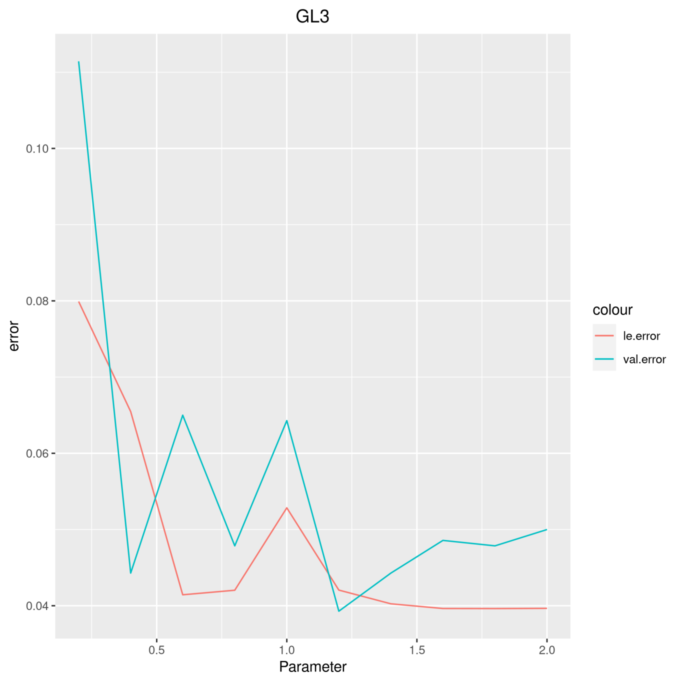

Figure 15: Error as a function of α

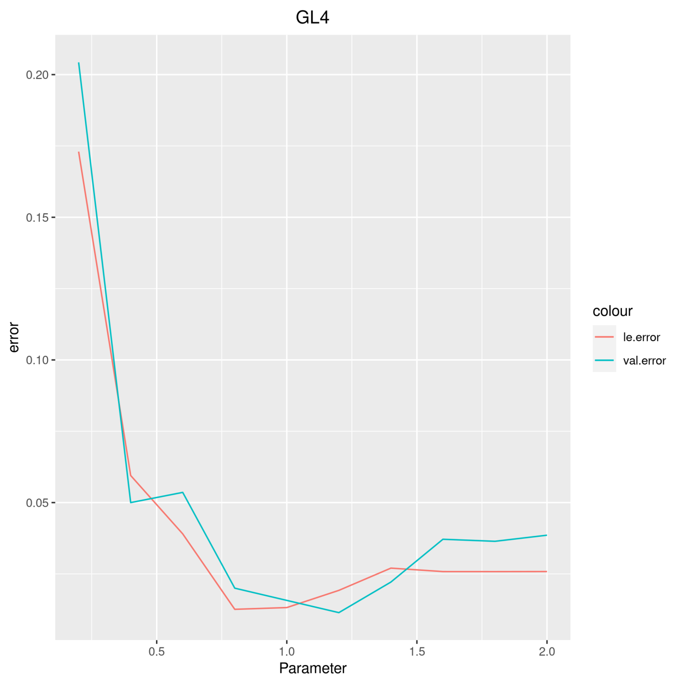

Figure 16: Error as a function of α

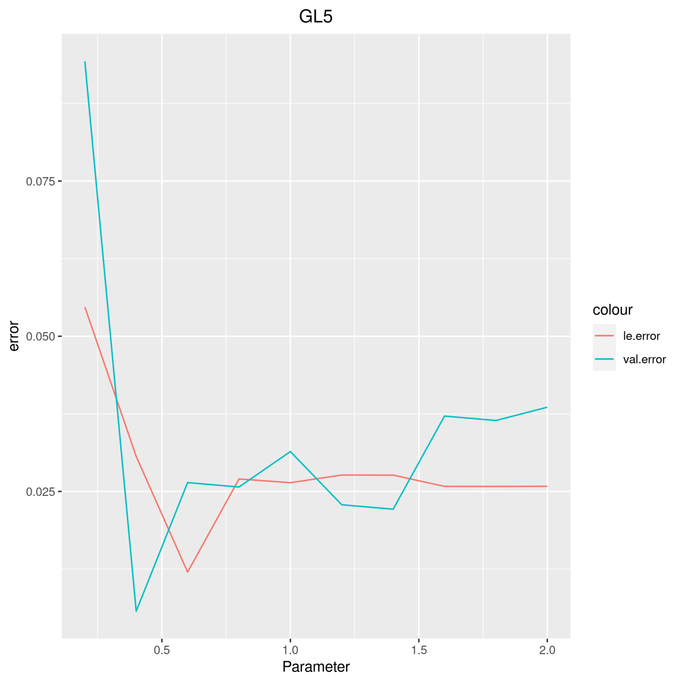

Figure 17: Error as a function of α

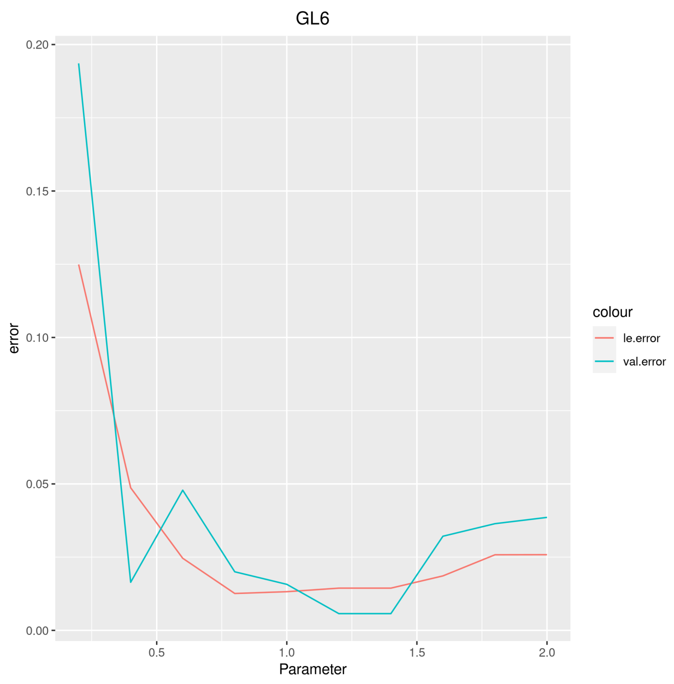

Figure 18: Error as a function of α

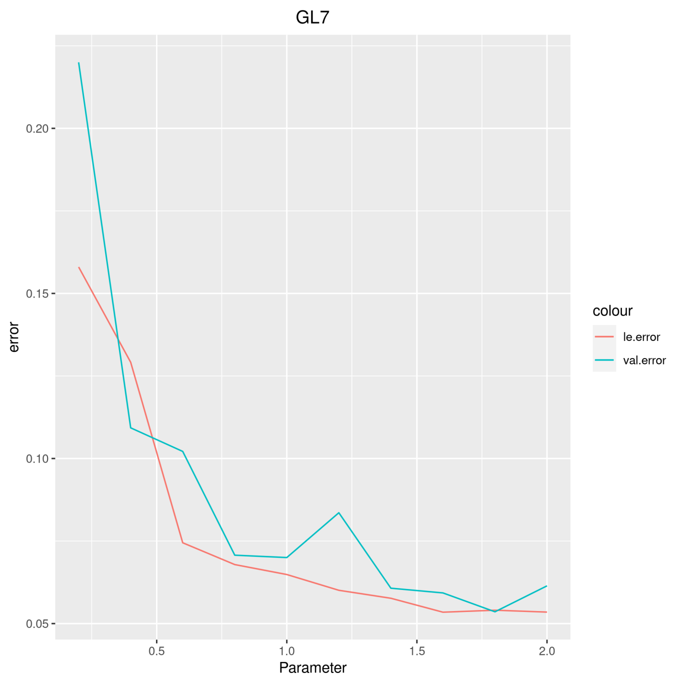

Figure 19: Error as a function of α

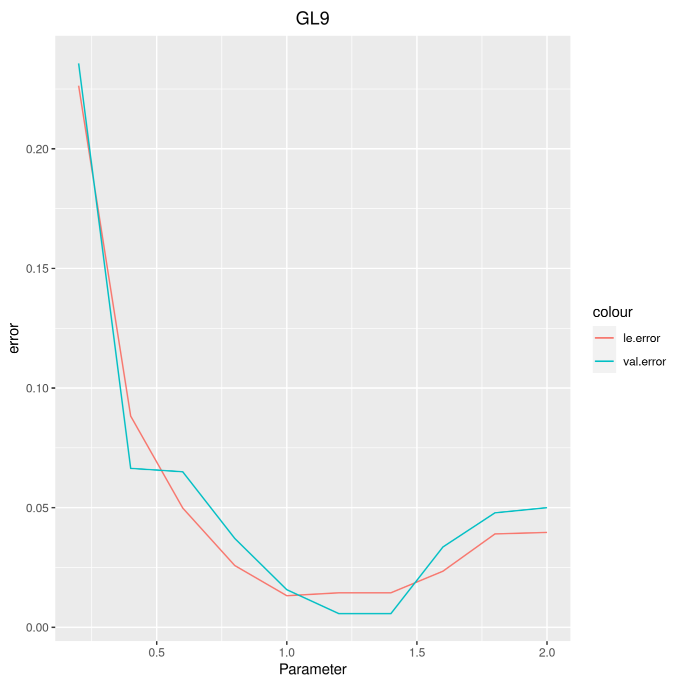

Figure 20: Error as a function of α

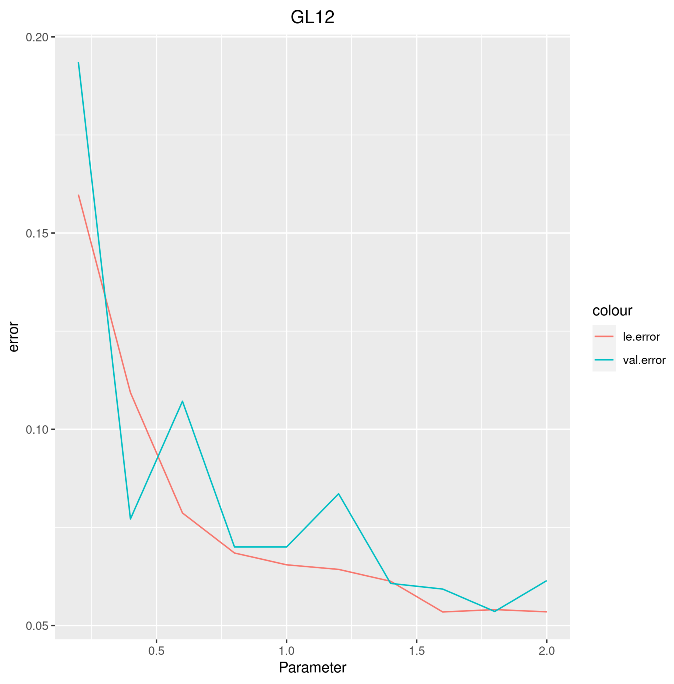

Figure 21: Error as a function of α

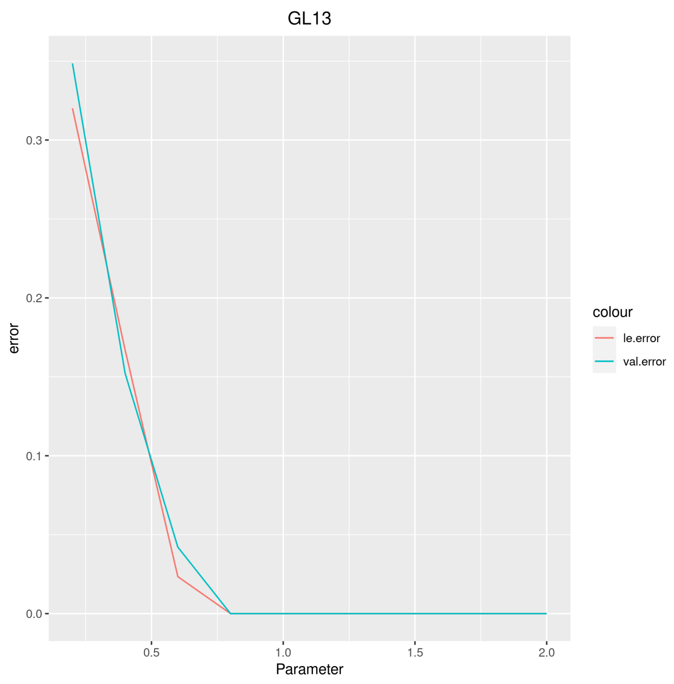

Figure 22: Error as a function of α

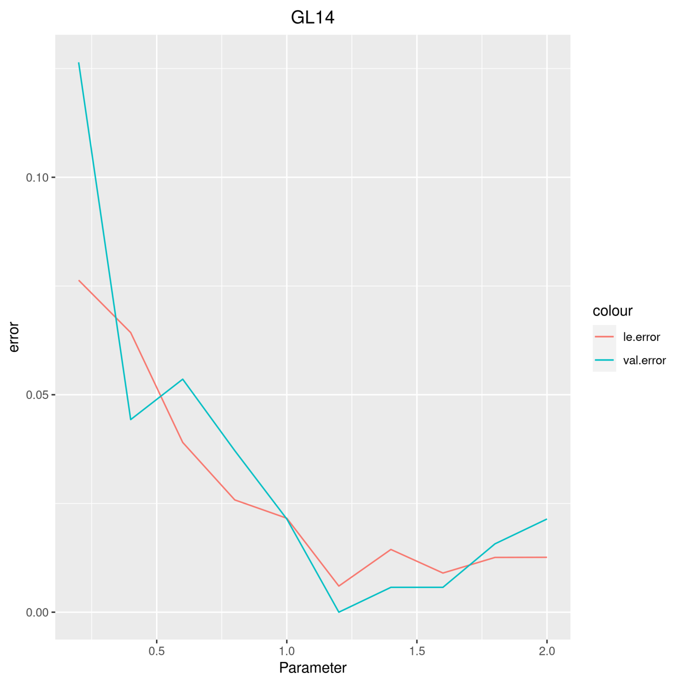

Figure 23: Error as a function of α

8.1.2 Gamma

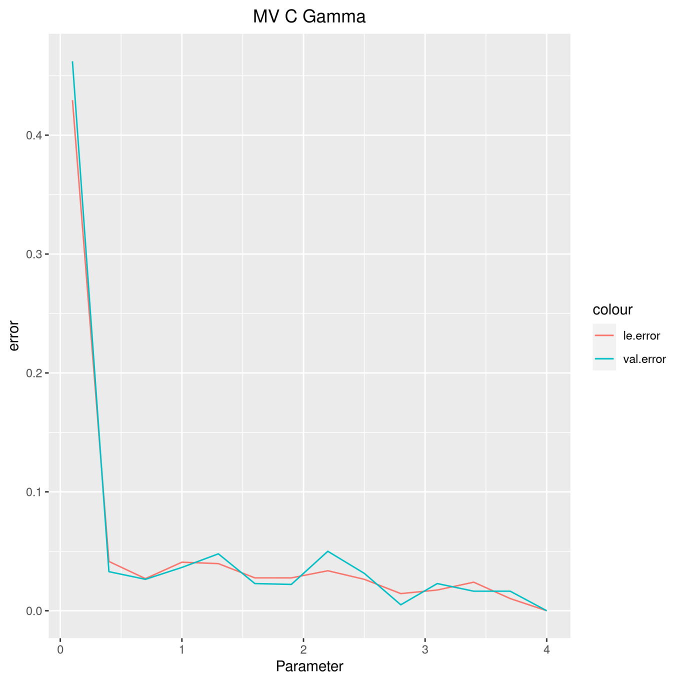

Figure 24: Error as a function of γ

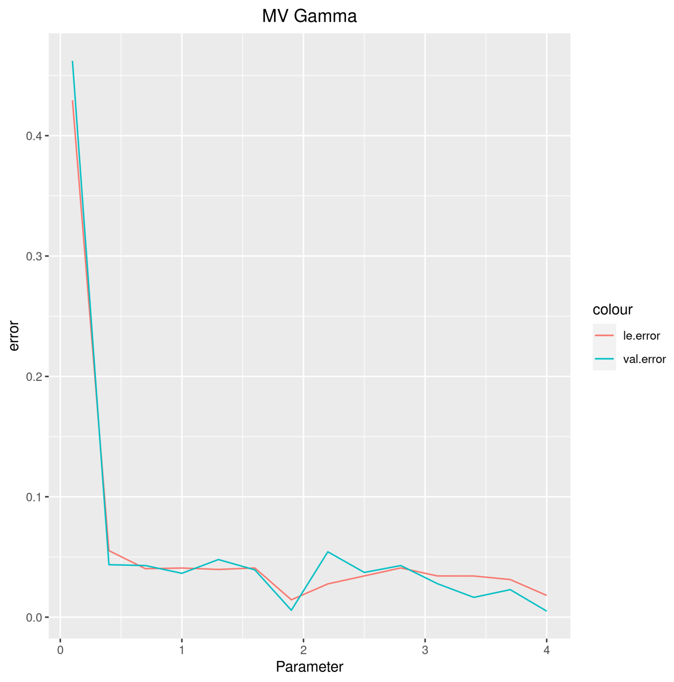

Figure 25: Error as a function of γ

References

[1] Lluis A. Belanche and Marco A. Villegas. *Kernel Functions for Categorical Variables with Application to Problems in the Life Sciences.* Technical University of Catalonia.

[2] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[3] Carlos Garcia Marquez. *Multivariate Kernel Functions for Categorical Variables.* Technical University of Catalonia, 2014.