STATISTICAL MACHINE LEARNING (SML2019)
1. COMPUTER LAB
SVM detector of generated domain names

VF

1 Introduction

The domain name systems (DNS) are nowadays frequently misused by a certain type of malware. For example, malware programs installed by attackers on a host computer try to use the DNS as a mean to establish connection with the Command and Control server in order to become a part of a botnet. The Command and Control server first registers a list of algorithmically generated domain names. The malware on an infected computer then queries the domain names generated by the same algorithm. The large number of generated domains prevents their manual detection. Examples of legitimate domain names and those generated by malware:

LEGITIMATE	GENERATED
google.com	atqgkfauhuaufm.com
facebook.com	vopydum.com
youtube.com	jgiugobavtyfsck.biz
yahoo.com	dhjopxgdetn.com
baidu.com	hhjjqjghfir.com
wikipedia.org	ldivjfkfamhnzjvbabqylvsij.info
qq.com	towngwvyjaebrtp.com
amazon.com	yyryxlgbgxsy.biz
live.com	mpwnnvqmxtnkv.ru
...	...

Our goal will be to design a classifier able to distinguish the generated domain names from the legitimate ones automatically. We will learn such classifier from a training set of examples by the Support Vector Machine algorithm. The training set $\mathcal{T} = \{(x^1, y^1), \ldots, (x^m, y^m)\} \in (\mathcal{X} \times \{+1, -1\})^m$ is composed of pairs of domain names and their labels. The input space $\mathcal{X} = \Sigma^*$ contains all strings that can be generated from a finite alphabet Σ. The label $y = +1$ will denote a generated malicious domain name while $y = -1$ will stand for a legitimate domain.

We will represent the input strings by the String Sub-sequence Kernel (SSK) [3]. The SSK $k_q \colon \Sigma^* \times \Sigma^* \to \mathbb{R}$ computes a dot product, $k_q(x, x') = \langle \boldsymbol{\phi}(x), \boldsymbol{\phi}(x') \rangle$, between two input strings embedded to a feature space via the map $\boldsymbol{\phi} \colon \Sigma^* \to \mathbb{R}^{|\Sigma|^q}$. The SSK compares two strings by means of the substrings of length q that may not be contiguous. The more substrings the input strings have in common, the more similar they are. The kernel is constructed such that less contiguous substrings contribute with less weight to the overall string similarity. The formal definition of the SSK is described in more details below.

This computer lab has been motivated by the paper [2].

2 Support Vector Machines

In the primal formulation, the SVM algorithm learns parameters $(\boldsymbol{w}, b) \in (\mathbb{R}^d, \mathbb{R})$ of the linear classifier

$$h(x; \boldsymbol{w}, b) = \operatorname{sign}\left(\langle \boldsymbol{w}, \boldsymbol{\phi}(x)\rangle + b\right), \tag{1}$$

by solving the following convex program

$$(\boldsymbol{w}^*, b^*) = \underset{\boldsymbol{w}\in\mathbb{R}^d, b\in\mathbb{R}}{\operatorname{argmin}} \left(\frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{m}\xi_i\right) \quad \text{s.t.} \quad \begin{array}{rcll} y^i(\langle \boldsymbol{w}, \boldsymbol{\phi}(x^i)\rangle + b) & \geq & 1-\xi_i\,, & i \in \{1,\ldots,m\}\,, \\ \xi_i & \geq & 0\,, & i \in \{1,\ldots,m\}\,, \end{array} \tag{2}$$

where $C > 0$ is a constant which controls the trade-off between the empirical error and the simplicity of the solution measured in terms of the L_2-norm.

The Lagrange dual of (2) reads

$$\boldsymbol{\alpha}^* = \underset{\boldsymbol{\alpha}\in[0,C]^m}{\operatorname{argmax}} \left(\sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\langle \boldsymbol{\phi}(x^i), \boldsymbol{\phi}(x^j)\rangle\, \alpha_i\, \alpha_j\, y^i\, y^j\right) \qquad \text{s.t.} \qquad \sum_{i=1}^{m}\alpha_i\, y^i = 0\,, \tag{3}$$

Having the dual solution $\boldsymbol{\alpha}^*$, the primal solution is obtained by

$$\boldsymbol{w}^* = \sum_{i=1}^{m}\alpha_i^*\, y^i\, \boldsymbol{\phi}(x^i) \qquad \text{and} \qquad b^* = y^i - \langle \boldsymbol{w}^*, \boldsymbol{\phi}(x^i)\rangle \quad \text{for any} \quad i \in \mathcal{I}_{\mathrm{SV}}^{\mathrm{b}}\,, \tag{4}$$

where $\mathcal{I}_{\mathrm{SV}}^{\mathrm{b}} = \{i \in \{1,\ldots,m\} \mid 0 < \alpha_i^* < C\}$. After substituting (4) to (1) we get

$$h(x; \boldsymbol{\alpha}^*, b^*) = \operatorname{sign}\left(\sum_{i=1}^{m} y^i \alpha_i^*\, \langle \boldsymbol{\phi}(x^i), \boldsymbol{\phi}(x)\rangle + b^*\right). \tag{5}$$

It is seen from (3), (4) and (5) that learning and evaluation of the classifier requires the inputs in terms of the dot products only. Therefore we can substitute $k(x^i, x^j)$ for any appearance of $\langle \boldsymbol{\phi}(x^i), \boldsymbol{\phi}(x^j)\rangle$ and use only the kernel functions that can be evaluated efficiently as we will see in the next section.

In your implementation you can use any library providing solver for (3). The SVM library should allow to use user-defined kernel matrix. You can also implement your own SVM solver but it is not required. We recommend the libSVM library [1] which provides simple interfaces for large number of programming languages including Matlab and Python. A code snipped in Matlab showing how to train SVM classifier with a user-defined kernel is here:

`http://cmp.felk.cvut.cz/cmp/courses/SSU/svm_dns/demo_libsvm.m`

3 String Sub-sequence Kernel

The formal definition of the SSK given in this section is adopted from the original paper [3]. The length of a string $s = s_1, \ldots, s_{|s|}$ is denoted by $|s|$. The concatenation of two strings s and t is denoted by st. The $s[i:j]$ is a substring $s_i, \ldots, s_j$ of the string s. The string u is a subsequence of s, if there exist indices $\boldsymbol{i} = (i_1, \ldots, i_{|u|})$, with $1 \leq i_1 < \cdots < i_{|u|} \leq |s|$, such that $u_j = s_{i_j}$, for $j = 1, \ldots, |u|$, or $u = s[\boldsymbol{i}]$ for short. The length $l(\boldsymbol{i})$ of subsequence $u = s[\boldsymbol{i}]$ is $l(\boldsymbol{i}) = i_{|u|} - i_1 + 1$.

The input strings from Σ^* are embedded to a feature space via $\boldsymbol{\phi}\colon \Sigma^* \to \mathbb{R}^{|\Sigma|^q}$. Each feature of $\boldsymbol{\phi}(s) = (\phi_u(s) \mid u \in \Sigma^q)$ is associated with one subsequence of length q. The feature value is defined as

$$\phi_u(s) = \sum_{\boldsymbol{i}\colon\, s(\boldsymbol{i})=u} \lambda^{l(\boldsymbol{i})}\,, \qquad u \in \Sigma^q\,,$$

where $\lambda \in (0, 1]$ is a decay parameter. The larger the length of the subsequence $u = s(\boldsymbol{i})$, the smaller contribution to the feature $\phi_u(x)$.

Let us consider subsequences of length $q = 2$ and strings "*cat*", "*car*", "*bat*" and "*bar*". The corresponding non-zero features are listed in the following table:

x	$\phi_{ca}(s)$	$\phi_{ct}(s)$	$\phi_{at}(s)$	$\phi_{ba}(s)$	$\phi_{bt}(s)$	$\phi_{cr}(s)$	$\phi_{ar}(s)$	$\phi_{br}(s)$
"*cat*"	λ^2	λ^3	λ^2	0	0	0	0	0
"*car*"	λ^2	0	0	0	0	λ^3	λ^2	0
"*bat*"	0	0	λ^2	λ^2	λ^3	0	0	0
"*bar*"	0	0	0	λ^2	0	0	λ^2	λ^3

For example, $\phi_{ca}(\text{"}cat\text{"}) = \lambda^2$ because the subsequence $u = \text{"}ca\text{"}$ appears in $s = \text{"}cat\text{"}$, indeed $s(\boldsymbol{i}) = u$ for $\boldsymbol{i} = (1, 2)$, and "*ca*" is of the length $l(\boldsymbol{i}) = 2$. However $\phi_{ct}(\text{"}cat\text{"}) = \lambda^3$ because the subsequence $u = \text{"}ct\text{"}$ appears in $s = \text{"}cat\text{"}$, that is $s(\boldsymbol{i}) = \text{"}ct\text{"}$ for $\boldsymbol{i} = (1, 3)$, but it has length $l(\boldsymbol{i}) = 3$ due to the skipped letter "*a*" in "*cat*".

The dot product of the feature vectors for strings s and t is

$$k_q(s,t) = \langle \boldsymbol{\phi}(s), \boldsymbol{\phi}(t) \rangle = \sum_{u\in\Sigma^q} \sum_{\boldsymbol{i}\colon\, u=s[\boldsymbol{i}]} \lambda^{l(\boldsymbol{i})} \sum_{\boldsymbol{j}\colon\, u=t[\boldsymbol{j}]} \lambda^{l(\boldsymbol{j})} = \sum_{u\in\Sigma^q} \sum_{\boldsymbol{i}\colon\, u=s[\boldsymbol{i}]} \sum_{\boldsymbol{j}\colon\, u=t[\boldsymbol{j}]} \lambda^{l(\boldsymbol{i})+l(\boldsymbol{j})}\,.$$

The dot products for $q = 2$ and strings "*cat*", "*car*", "*bat*", "*bar*" are as follows:

$k_2(s,t)$	"*cat*"	"*car*"	"*bat*"	"*bar*"
"*cat*"	$2\lambda^4 + \lambda^6$	λ^4	λ^4	0
"*car*"	λ^4	$2\lambda^4 + \lambda^6$	0	λ^4
"*bat*"	λ^4	0	$2\lambda^4 + \lambda^6$	λ^4
"*bar*"	0	λ^4	λ^4	$2\lambda^4 + \lambda^6$

In order to make the similarity less dependent on the length of the input string, it is recommended to use the L_2-normalized kernel

$$\hat{k}_q(s,t) = \frac{k_q(s,t)}{\sqrt{k_q(s,s)}\sqrt{k_q(t,t)}}\,.$$

A naive computation of the kernel value requires $\mathcal{O}(|\Sigma|^q)$ time and space since this is the number of features involved. An efficient algorithm to evaluate the kernel uses an auxiliary function

$$k'_i(s,t) = \sum_{u\in\Sigma^i} \sum_{\boldsymbol{i}\colon\, u=s[\boldsymbol{i}]} \sum_{\boldsymbol{j}\colon\, u=t[\boldsymbol{j}]} \lambda^{|s|+|t|-i_1-j_1+2}\,, \qquad i \in \{1, \ldots, q-1\}$$

The auxiliary kernel function $k'_i(s,t)$ differs from the original kernel $k_i(s,t)$ just in the way how it counts the length of the subsequences. In particular, $k'_i(s,t)$ counts the length from the beginning of the sequence till the end of the strings instead of using the position of the last symbol. For example, the auxiliary kernel counts the length of the subsequence "*ca*" in the string "*cat*" to be 3 instead of 2.

Using the auxiliary kernel, the SSK can be computed by the following recursive procedure:

$$
\begin{aligned}
k'_0(s,t) &= 1 \text{ , for all } s,t \\
k'_i(s,t) &= 0 \text{ , if } \min\{|s|,|t|\} < i \\
k_i(s,t) &= 0 \text{ , if } \min\{|s|,|t|\} < i \\
k'_i(sx,t) &= \lambda k'_i(s,t) + \sum_{j:\, t_j = x} k'_{i-1}(s, t[1,\ldots,j-1])\lambda^{|t|-j+2}\,, \qquad i \in \{1,\ldots,q-1\} \\
k_q(sx,t) &= k_q(s,t) + \sum_{j:\, t_j = x} k'_{q-1}(s, t[1,\ldots,j-1])\lambda^2
\end{aligned}
$$

4 Data

The data can be downloaded from this link

`http://cmp.felk.cvut.cz/cmp/courses/SSU/svm_dns/dns_data.zip`

The zip file contains:

`dns_data.mat` ... training, validation and test examples stored in Matlab format.

`{trn,tst,val}_malware.txt` ... malicious examples stored in a plain text file.

`{trn,tst,val}_legit.txt` ... contains legitimate examples stored in a plain text file.

The examples in the MAT file and the text files are identical. The two formats are used just for your convenience (Matlab vs. Python programmers). The examples are split randomly into three parts: 1000 training, 500 validation and 2000 test examples. The training examples serve for learning the parameters $\boldsymbol{\alpha}$ and b of the kernel SVM classifier (5). The validation examples should be used for selection of the best value of the hyper-parameter C. The test examples are intended for the final evaluation of the SVM classifier with the best C.

For those students who do not manage to implement the SSK kernel we provide precomputed kernel matrices:

K_{trn} $[1000 \times 1000]$ dot products between training examples.

K_{val} $[500 \times 1000]$ dot products between validation and training examples.

K_{tst} $[2000 \times 1000]$ dot products between test and training examples.

The kernel values were computed using the L_2-normalization, $q = 2$ and $\lambda = 0.4$. The kernel matrices can be downloaded from this link

`http://cmp.felk.cvut.cz/cmp/courses/SSU/svm_dns/dns_data_kernel.zip`

The zip file contains:

`dns_data_kernel.mat` ... MAT file containing all kernel matrices together with the labels.

`{trn,val,tst}_kernel_mat.svmlight` ... kernel matrices with corresponding labels stored in SVM^{light} format. The SVM^{light} format is plain text where each line starts with the label ($+1 or -1$) followed by a list of indices and values of the non-zero elements of the corresponding row of the kernel matrix:

```
<label> <index1>:<value1> <index2>:<value2> ...
```

Hence the file has as many lines as the number of rows of the corresponding kernel matrix. The Matlab interface of the libSVM provides a function to load the labels and the kernel matrix by

```
[labels, kernel_matrix] = libsvmread('{trn,val,tst}_kernel_mat.svmlight')
```

5 Task assignment

Assignment 1 (6 points) *Train the kernel SVM classifier for a set of regularization constants* $C \in \{0.01, 0.1, 1, 10, 100\}$. *Use the* L_2*-normalized SSK kernel with* $q = 2$ *and* $\lambda = 0.4$. *Create a table which will contain the training error, the validation error and the number of support vectors for each* $C \in \{0.01, 0.1, 1, 10, 100\}$. *In addition, display the training errors and the validation error as a function of* C *in a graph (use logarithmic x-axis). By errors we mean the empirical risk with* $0/1$*-loss estimated on the corresponding part of the examples.*

Remark: You are allowed to use the pre-computed kernel matrices.

Assignment 2 (4 points) *Select the best regularization constant* C *based on the minimal validation error. Report the best constant* C *and the test error* $R_{\mathcal{S}^l}(h)$ *of the corresponding SVM classifier. Compute the minimal value of* ε *such that the true classification error is in the interval* $(R_{\mathcal{S}^l}(h) - \varepsilon, R_{\mathcal{S}^l}(h) + \varepsilon)$ *with the probability* 99% *at least.*

Assignment 3 (5 bonus points) *Implement a function which computes the SSK kernel. For sequence length* $q = 3$ *(note it is different length than in Assignment 1) and the decay parameter* $\lambda = 0.4$, *compute a kernel matrix describing similarity between the following strings: "google.com", "facebook.com", "atqgkfauhuaufm.com" and "vopydum.com". Repeat the computation for* L_2*-normalized SSK. Report the obtained kernel matrices in the document (in total 2 matrices of size* $\mathbb{R}^{4\times4}$*). The implemented function should have the following syntax:*

```
k = subseq_kernel( str1, str2, q, lambda)
```

Remark: Assignment 3 is not obligatory to gain full number of points from this lab (that is 10). However, if you solve it you will gain 5 bonus points.

Hint: It is recommended not to use recursive functions in Matlab. Rather implement the recurrent formulas using embedded loops.

References

[1] Chang Chih-Chung and Lin Chih-Jen. Libsvm: `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`.

[2] Haddadi F. and Zincir-Heywood A.N. Analyzing string format-based classifiers for botnet detection: GP and SVM. In *IEEE Congress on evolutionary Computation*, 2013.

[3] Lodhi H., Saunders C., Shawe-Taylor J., Cristianini N., and Watkins C. Text classification using string kernels. *Journal of Machine Learning Research*, 2002.