第 9 章 支持向量机算法

支持向量机是分类算法里面使用范围最为广泛的算法之一，甚至可以自豪的说，没有之一，它被广泛应用在生活中的众多领域，基本所有的分类问题，尤其是二分类问题都可以先用它试下，众多机器学习的教材甚至都把支持向量机作为第一个介绍的算法，可见其重要性。

本章 9.1 节将结合红蓝求的故事介绍生活中的支持向量机算法。

本章 9.2 节和 9.3 节将介绍支持向量机的基本概念，并给出基本的使用方式。

本章 9.4 节将介绍如何使用支持向量机来检测 XSS。

本章 9.5 节将介绍如何使用支持向量机来区分僵尸网络的 DGA 家族。

9.1 生活中的支持向量机

www.reddit.com 上面有个非常有趣的帖子，Lvhhh 用白话文做了诠释，非常形象。

某日，见蓝球红球于一桌欲分之。

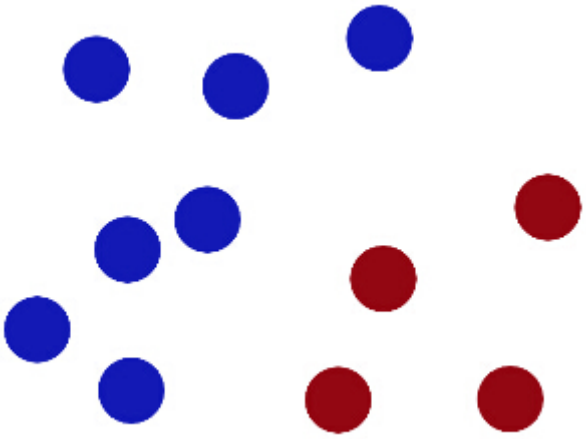

图9-1 红蓝球故事

插一筷子于蓝红球之间则蓝红球可分。

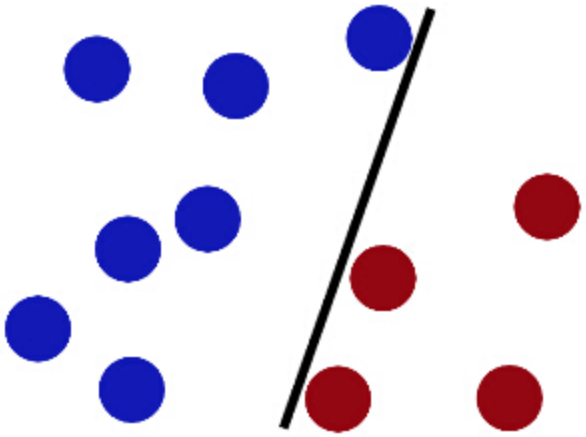

图9-2 红蓝球故事

不料，随着球之增多，一红球出界毁吾之分割。可惜可气。

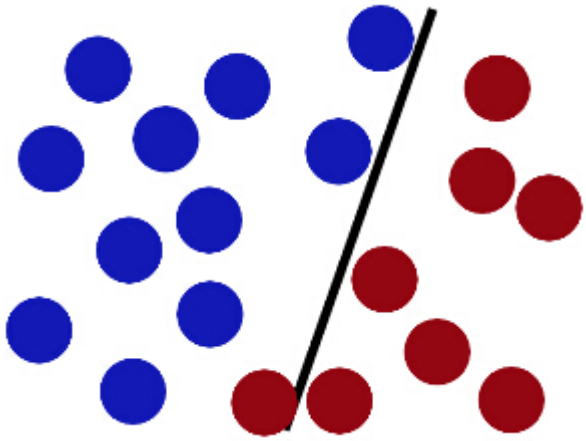

图9-3 红蓝球故事

不服，遂变化筷子方向则又可分红蓝球也。

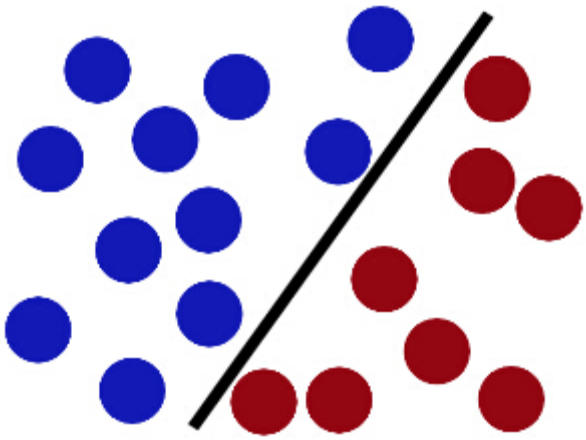

图9-4 红蓝球故事

终有体会，欲合理分清红蓝之球，必使得近处红蓝球于筷子越远越好。

他日，又偶遇如下一堆红蓝球，吾又欲分之

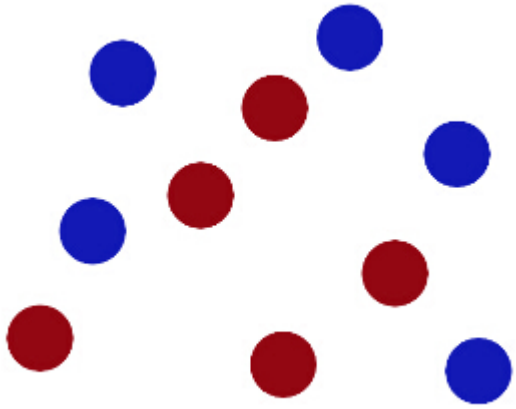

图9-5 红蓝球故事

拿筷子比划半天无从分离，百思不得其解。大怒，猛一拍桌。

见桌上之球于空中仿佛有可分之势，蓝上红下。大喜，顺势抽一张纸隔于蓝红球之间，则蓝红之球可分。

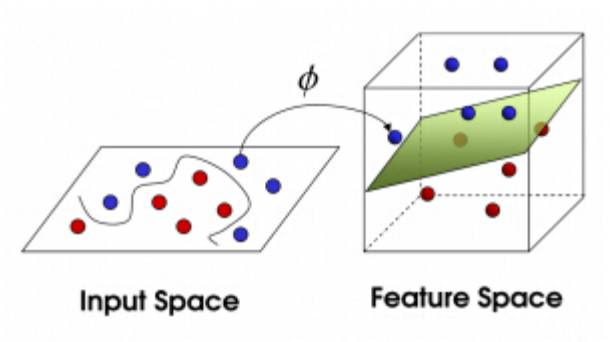

图9-6 红蓝球故事

遂可得，若桌面上不可分（2 维），则拍桌，将球腾空而起（3 维），则可分之。

9.2 支持向量机概述

SVM（Support Vector Machine，支持向量机）是机器学习领域使用最广泛的算法之一，通常用来进行模式识别、分类以及回归分析，它特别适合安全世界里面的非黑即白，所以我们重点介绍分类相关的知识。

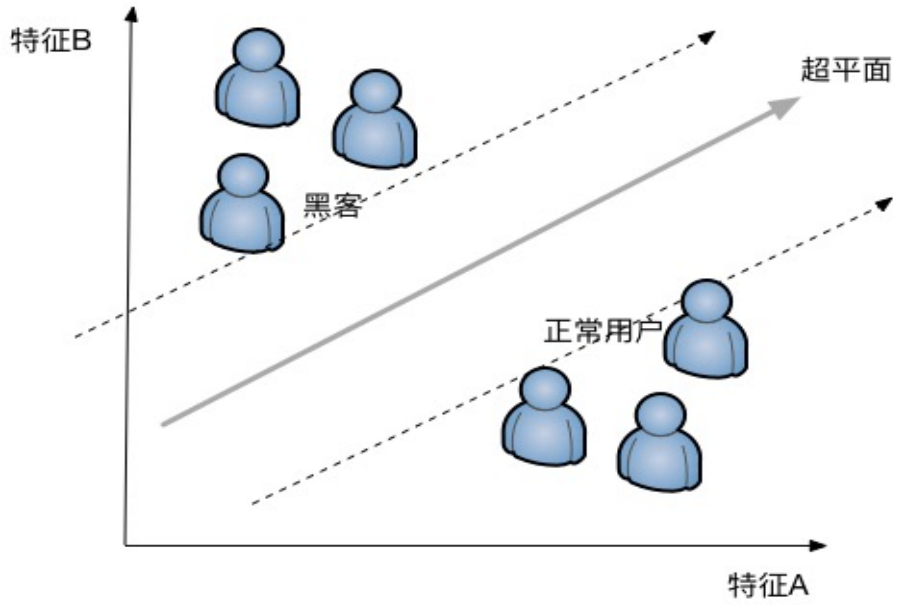

图9-7 SVM 原理图

假设只有二维的特征向量，我们需要解决一个分类问题，需要通过将正常用户和黑客区分开来，如果确实可以通过一条直线区分，那么这个问题成为可以线性可区分（linear separable），如果不行则成为不可线性区分（linear inseparable）。讨论最简单的情况，假设这个分类问题是可以线性区分的，那么这个区分的直线成为超平面，距离超平面最近的样本成为支持向量（Supprot Verctor）。

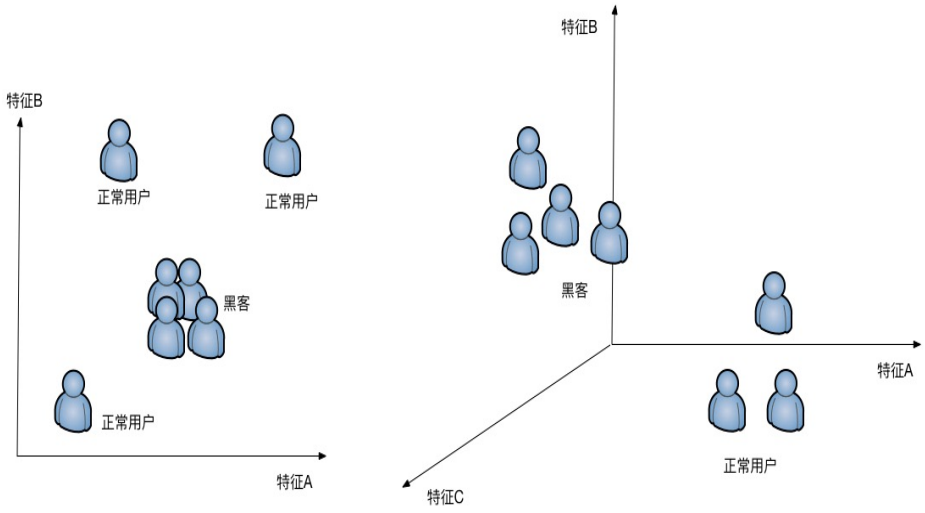

图9-8 SVM 从二维平面升级到三维平面

如上图，对于不可线性区分的情况，需要升级到更高的平面进行区分，比如二维平面搞不定就需要升级到三维平面来区分，这个升级就需要依靠核函数。SVM 通过一个非线性映射，把样本空间映射到一个高维乃至无穷维的特征空间中（Hilbert 空间），使得在原来的样本空间中非线性可分的问题转化为在特征空间中的线性可分的问题。简单地说，就是升维和线性化。升维，就是把样本

向高维空间做映射，一般情况下这会增加计算的复杂性，甚至会引起“维数灾难”，因而人们很少问津。但是作为分类、回归等问题来说，很可能在低维样本空间无法线性处理的样本集，在高维特征空间中却可以通过一个线性超平面实现线性划分（或回归）．一般的升维都会带来计算的复杂化，SVM 方法巧妙地解决了这个难题：应用核函数的展开定理，就不需要知道非线性映射的显式表达式；由于是在高维特征空间中建立线性学习机，所以与线性模型相比，不但几乎不增加计算的复杂性，而且在某种程度上避免了“维数灾难”．这一切要归功于核函数的展开和计算理论。

选择不同的核函数，可以生成不同的 SVM，常用的核函数有以下 4 种：

- 线性核函数 K(x,y)=x·y
- 多项式核函数 K(x,y)=[(x·y)+1]^d
- 径向基函数 K(x,y)=exp(-|x-y|^2/d^2）
- 二层神经网络核函数 K(x,y)=tanh(a(x·y)+b)

9.3 示例：hello world！支持向量机

我们先演示支持向量机的基础使用，完整演示代码请见本书 Github 上的 9-1.py。

导入库文件

```
print(__doc__)
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
```

创建 40 个随机点

```
np.random.seed(0)
X = np.r_[np.random.randn(20, 2) - [2, 2], np.random.randn(20, 2) + [2, 2]]
Y = [0] * 20 + [1] * 20
# fit the model
clf = svm.SVC(kernel='linear')
clf.fit(X, Y)
```

构造超平面

```
w = clf.coef_[0]
```

```
a = -w[0] / w[1]

xx = np.linspace(-5, 5)

yy = a * xx - (clf.intercept_[0]) / w[1]

# plot the parallels to the separating hyperplane that pass through the

# support vectors

b = clf.support_vectors_[0]

yy_down = a * xx + (b[1] - a * b[0])

b = clf.support_vectors_[-1]

yy_up = a * xx + (b[1] - a * b[0])
```

调用 matplotlib 画图

```
plt.plot(xx, yy, 'k-')

plt.plot(xx, yy_down, 'k--')

plt.plot(xx, yy_up, 'k--')

plt.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1],

           s=80, facecolors='none')

plt.scatter(X[:, 0], X[:, 1], c=Y, cmap=plt.cm.Paired)

plt.axis('tight')

plt.show()
```

运行代码

```
localhost:code maidou$ python 9-1.py

None

localhost:code maidou$
```

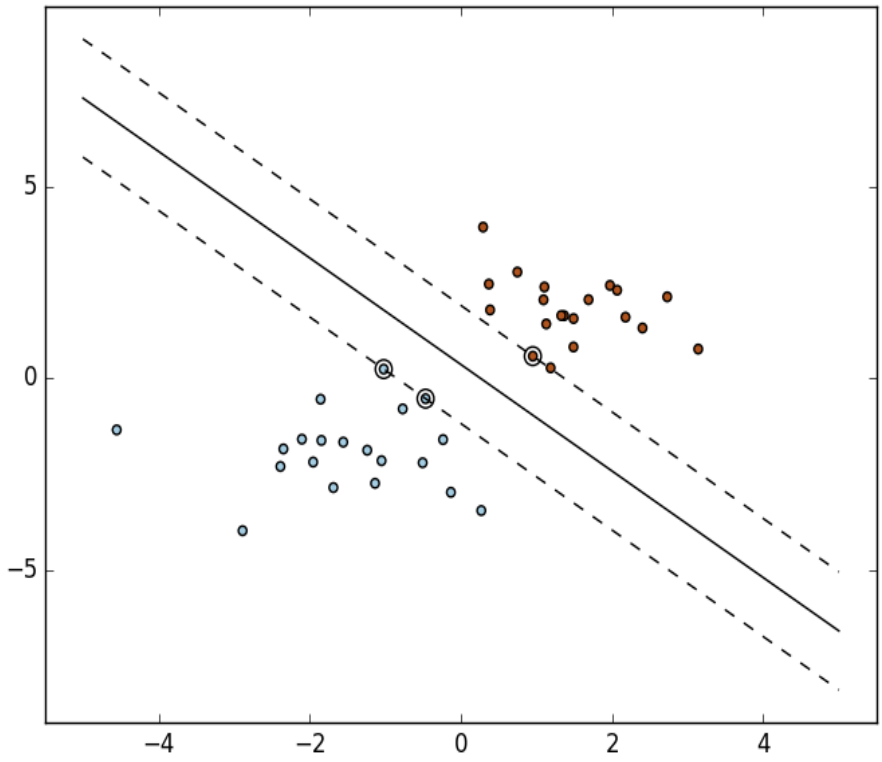

图9-9 SVM hello word 代码

9.4 示例：使用支持向量机识别 XSS

下面以常见的 XSS 检测来说明下 SVM 的简单应用。完整演示代码请见本书 Github 上的 9-2.py。

9.4.1 处理流程

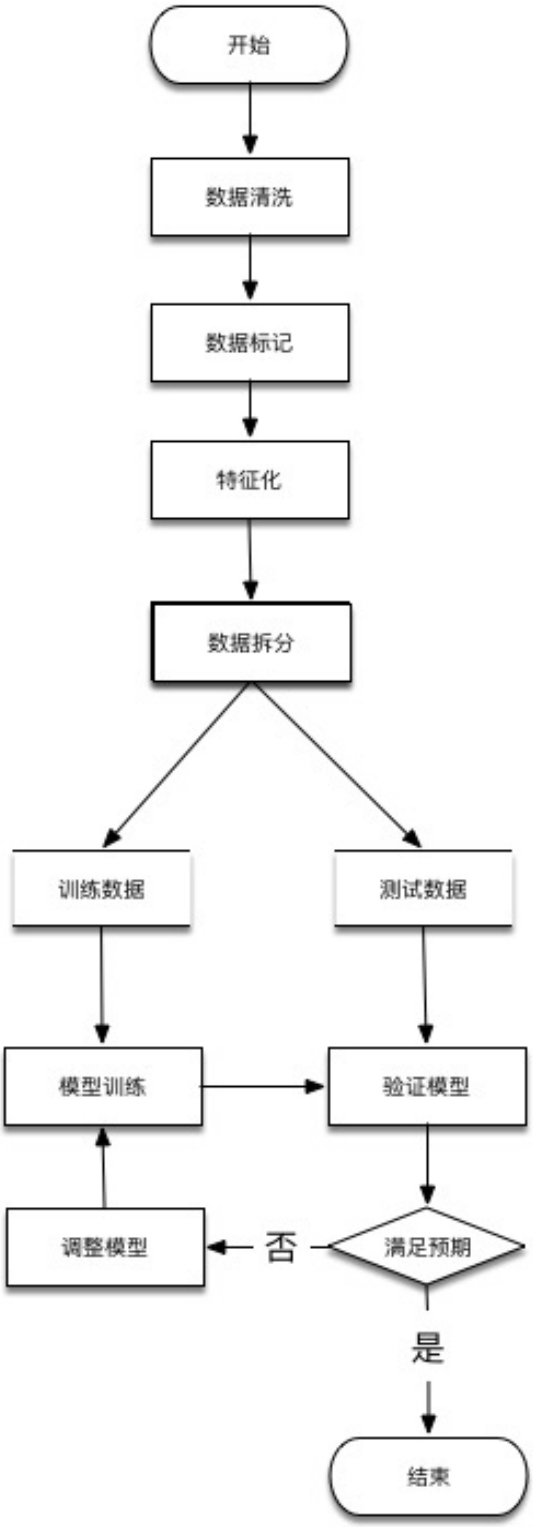

图9-10 数据处理流程

9.4.2 数据搜集&数据清洗

由于我们的例子比较简单，把上述两个步骤合并即可，准备数量相等的正常 web 访问日志和 XSS 攻击的 web 日志，最简单的方法是参考我以前的文章《基于 WAVSEP 的靶场搭建指南》，使用 WVS 等扫描器仅扫描 XSS 相关漏洞即可获取 XSS 攻击的 web 日志。

9.4.3 特征化

实践中数据搜集&数据清洗是最费时间的，特征化是最烧脑的，因为世界万物是非常复杂的，具有很多属性，然而机器学习通常只能理解数字向量，这个从现实世界的物体转变成计算世界的数字的过程就是特征化，也叫向量化。比如要你特征化你前女友，你总不能说漂亮、温柔这些词，需要最能代表她的特点的地方进行数字化，下面是一个举例：

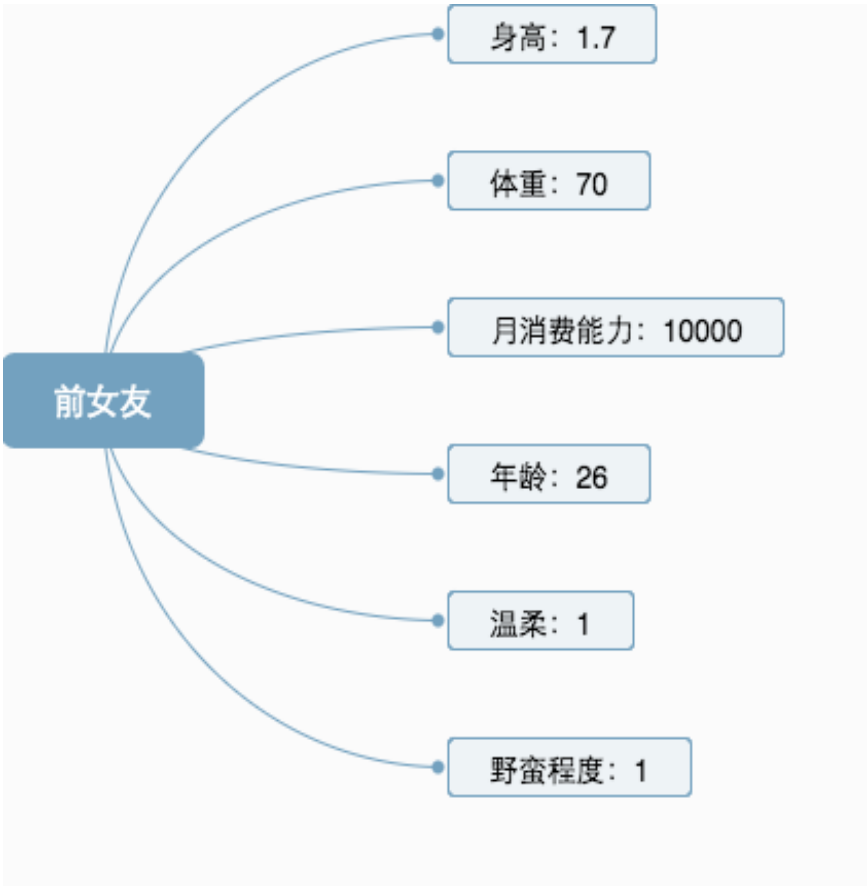

图9-11　现实生活中将事物向量化的例子

你会发现各个向量之间的数据范围差别很大，一个月消费能力可能就干掉其他特征对结果的影响了，虽然现实生活中这个指标确实影响很大，但是不足以干掉其他全部特征，所以我们还需要对

特征进行标准化，常见的方式为：

- 标准化
- 均方差缩放
- 去均值

回到 XSS 的问题上，我们需要针对 web 日志进行特征化，以下特征是个举例：

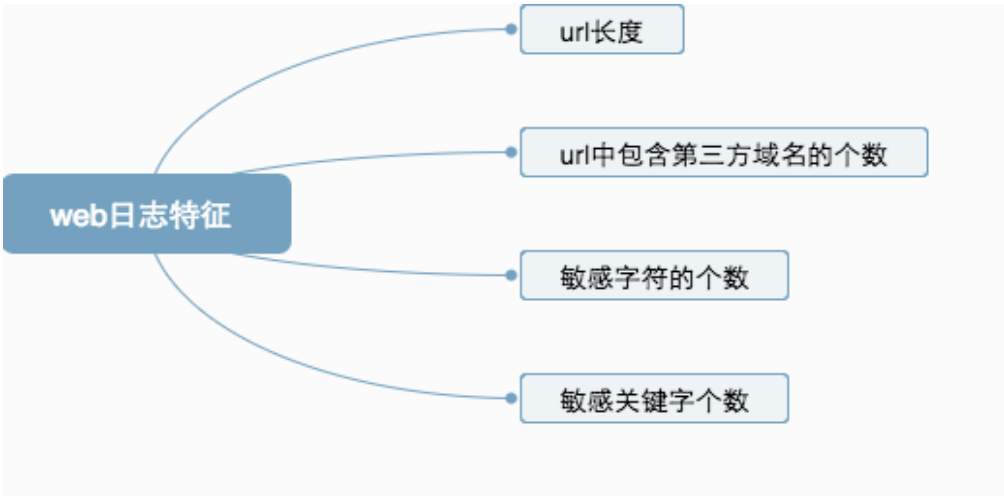

图9-12　　web 日志特征举例

特征提取的示例代码如下：

```
   def get_len(url):
         return len(url)
   def get_url_count(url):
         if re.search('(http://)|(https://)', url, re.IGNORECASE) :
                  return 1
         else:
                  return 0
   def get_evil_char(url):
         return len(re.findall("[<>,\'\"/]", url, re.IGNORECASE))
   def get_evil_word(url):
         return
len(re.findall("(alert)|(script=)(%3c)|(%3e)|(%20)|(onerror)|(onload)|(eval)|
(src=)|(prompt)",url,re.IGNORECASE))
```

数据标准化使用如下代码即可：

```
min_max_scaler = preprocessing.MinMaxScaler()
x_min_max=min_max_scaler.fit_transform(x)
```

9.4.4 数据打标

XSS 标记为 1，正常访问标记为 0。

9.4.5 数据拆分

这一步是为了随机把数据区分成训练组和测试组，通常直接使用 cross_validation.train_test_split 即可，通常使用 40%作为测试样本，60%作为训练样本，这个比例可以根据自己需要调节。

```
x_train, x_test, y_train, y_test = cross_validation.train_test_split(x,y,
test_size=0.4, random_state=0)
```

9.4.6 数据训练

使用 scikit-learn 的 SVM 模型即可，SVM 用于分类的模型称为 SVC，我们使用最简单的核函数 linear

```
clf = svm.SVC(kernel='linear', C=1).fit(x, y)
joblib.dump(clf,"xss-svm-200000-module.m")
```

9.4.7 模型验证

通过加载训练后的模型，针对测试集合进行预测，将预测结果与打标结果比对即可。

```
clf=joblib.load("xss-svm-200000-module.m")
y_test=[]
y_test=clf.predict(x)
print metrics.accuracy_score(y_test, y)
```

测试环节我们在一个各有 200000 个样本黑白模型上训练，在一个各有 50000 个样本的黑白测试集上校验，任何黑白预测错都判断为错误，最后运行结果准确度率为 80%，对于机器学习而言，

仅依靠模型优化，这个比例已经很高了。通过在更大的数据集合上进行训练（比如大型 CDN&云 WAF 集群的日志），进一步增加特征个数以及增加后面环节的自动化或者半自动化的验证，可以进一步提高这个比例，最后准确率我们做到了 90%以上。下图是特征扩展的举例，大家可以根据实际情况增加。

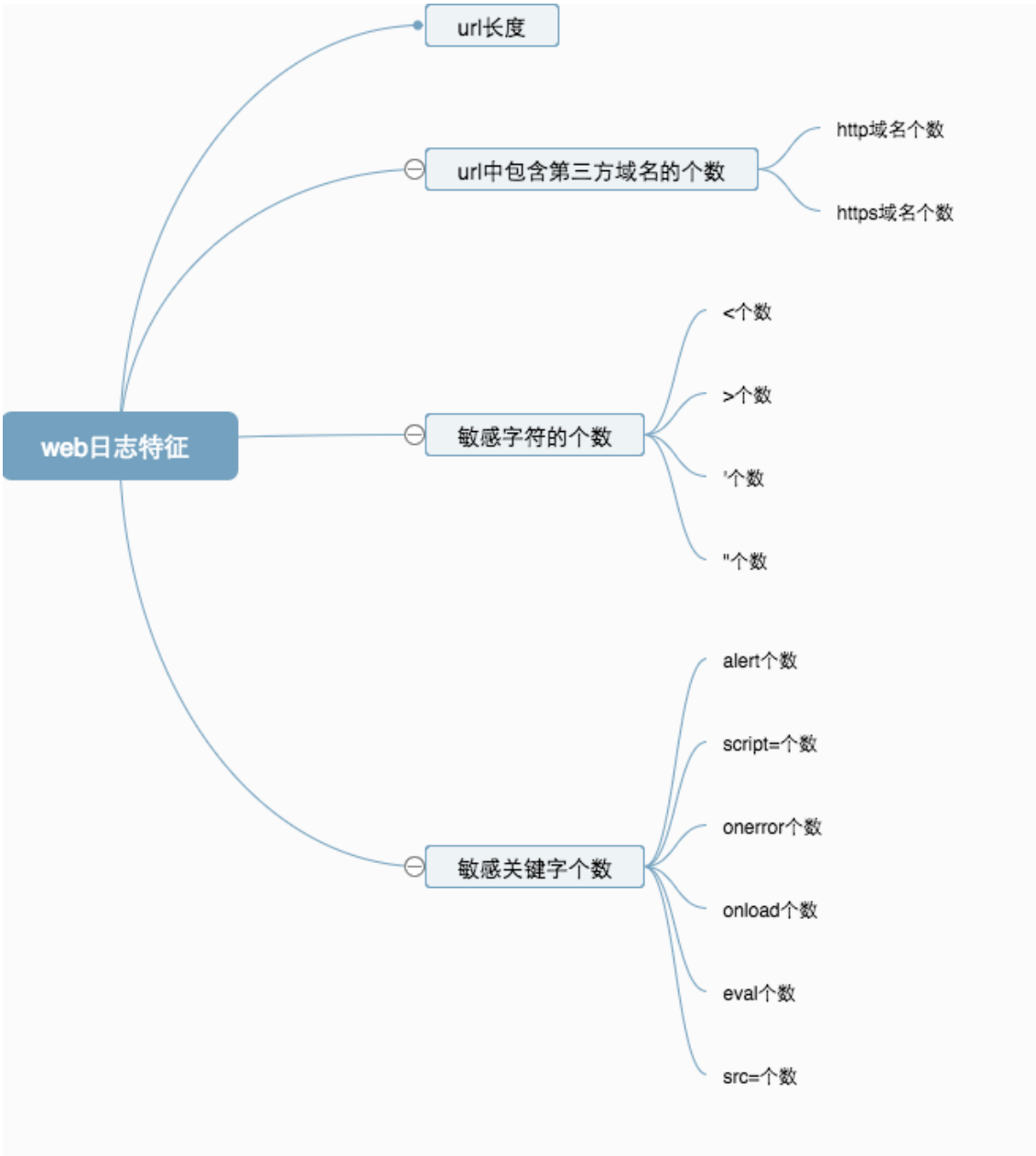

图9-13　　web 日志特征

9.4.8　异常数据

通过 SVM 我们识别出了异常数据，经过人工确认，除了变形的 XSS 以外，还有不少其他攻击

行为，由于测试时只打开了 XSS 的规则签名，所以其他攻击行为没有拦截，也进入了白样本，举例如下：

```
   /.|./.|./.|./.|./.|./.|./.|./.|./.|./.|./.|./windows/win.ini
   /index.php?op=viewarticle&articleid=9999/**/union/**/select/**/1331908730,
1,1,1,1,1,1,1--&blogid=1
   /index.php?go=detail&id=-99999/**/union/**/select/**/0,1,concat(1331919200
,0x3a,512612977),3,4,5,6,7,8,9,10,11,12,13,14,15,16
   /examples/jsp/num/.|./.|./.|./.|./.|./.|./.|./.|./.|./.|./.|./.|./.|./windows/
win.ini
   /cgi-bin/modules/tinymce/content_css.php?templateid=-1/**/union/**/select/
**/1331923205,1,131076807--
   /manager/ajax.php?rs=__exp__getfeedcontent&rsargs[]=-99 union select
1161517009,2,1674610116,4,5,6,7,8,9,0,1,2,3 --
```

我们推测，机器从 XSS 样本里面学习的攻击特征可能部分覆盖了 SQL 注入等带有代码注入特性的攻击的特点导致的。

9.5 示例：使用支持向量机区分僵尸网络 DGA 家族

僵尸网络为了躲避域名黑名单，通常会使用 DGA 技术动态生成域名，通过 DGA 域名的不同特征，可以尝试识别不同的家族群。我们以常见的 cryptolocker 和 post-tovar-goz 两个僵尸网络家族为例子。完整演示代码请见本书 Github 上的 9-3.py。

9.5.1 数据搜集&数据清洗

实验阶段，我们搜集了如下数据：

- 1000 个 cryptolocker 域名
- 1000 个 post-tovar-goz 域名
- alexa 前 1000 域名

DGA 文件格式如下：

```
   xsxqeadsbgvpdke.co.uk,Domain used by Cryptolocker - Flashback DGA for 13 Apr
2017,2017-04-13,http://osint.bambenekconsulting.com/manual/cl.txt
```

从 DGA 文件中提取域名数据：

```
   def load_dga(filename):
    domain_list=[]
    #xsxqeadsbgvpdke.co.uk,Domain used by Cryptolocker - Flashback DGA for 13 Apr
2017,2017-04-13,
    # http://osint.bambenekconsulting.com/manual/cl.txt
    with open(filename) as f:
        for line in f:
            domain=line.split(",")[0]
            if domain >= MIN_LEN:
                domain_list.append(domain)
    return  domain_list
```

alexa 文件使用 CSV 格式保存域名的排名和域名，提取数据方式如下：

```
   def load_alexa(filename):
         domain_list=[]
         csv_reader = csv.reader(open(filename))
         for row in csv_reader:
                  domain=row[1]
                  if domain >= MIN_LEN:
                  domain_list.append(domain)
         return domain_list
```

9.5.2 特征化

1 元音字母个数

正常人通常在取域名的时候，都会偏向选取”好读”的几个字母组合，抽象成数学可以理解的语

言，就是英文的元音字母比例会比较高。DGA 生成域名的时候，由于时间因素是随机因素，所以元音字母这方面的特征不明显。下面我们通过数据分析来验证我们的想法。

读取 alexa 域名数据：

```
x1_domain_list = load_alexa("../data/top-1000.csv")
```

计算元音字母的比例：

```
def get_aeiou(domain_list):
 x=[]
 y=[]
 for domain in domain_list:
    x.append(len(domain))
    count=len(re.findall(r'[aeiou]',domain.lower()))
    count=(0.0+count)/len(domain)
    y.append(count)
 return x,y
```

分别获取两个僵尸网络 DGA 域名数据以及 alexa 域名数据并计算元音字母比例：

```
x1_domain_list = load_alexa("../data/top-1000.csv")
x_1,y_1=get_aeiou(x1_domain_list)
x2_domain_list = load_dga("../data/dga-cryptolocke-1000.txt")
x_2,y_2=get_aeiou(x2_domain_list)
x3_domain_list = load_dga("../data/dga-post-tovar-goz-1000.txt")
x_3,y_3=get_aeiou(x3_domain_list)
```

以域名长度为横轴，元音字母比例为纵轴作图：

```
fig,ax=plt.subplots()
ax.set_xlabel('Domain Length')
ax.set_ylabel('AEIOU Score')
ax.scatter(x_3,y_3,color='b',label="dga_post-tovar-goz",marker='o')
ax.scatter(x_2, y_2, color='g', label="dga_cryptolock",marker='v')
ax.scatter(x_1, y_1, color='r', label="alexa",marker='*')
```

```
ax.legend(loc='best')

plt.show()
```

分析图表，不同家族之间具有明显聚合效果，正常域名与 DGA 之间具有一定的区分性。

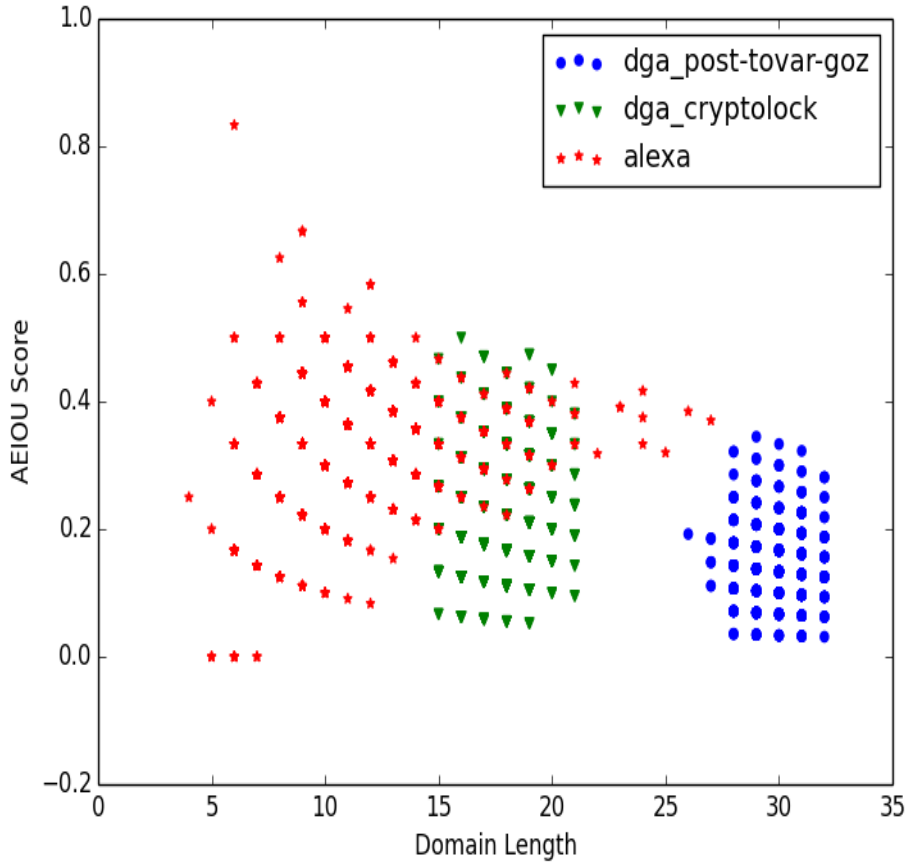

图9-14　　元音字母比例分布图

2　唯一数字字母个数

唯一的字母数字个数指的是域名中去掉重复的字母和数字后的个数，比如：

- baidu 的个数为 5
- facebook 个数为 7
- google 的个数为 4

唯一的字母数字个数与域名长度的比例，从某种程度上反应了域名字符组成的统计特征。计算唯一的字母和数字个数可以使用 python 的 set 数据结构：

```
def get_uniq_char_num(domain_list):
    x=[]
    y=[]
    for domain in domain_list:
        x.append(len(domain))
        count=len(set(domain))
        count=(0.0+count)/len(domain)
        y.append(count)
    return x,y
```

分别获取两个僵尸网络 DGA 域名数据以及 alexa 域名数据并计算元音字母比例：

```
x1_domain_list = load_alexa("../data/top-1000.csv")
x_1,y_1=get_uniq_char_num(x1_domain_list)
x2_domain_list = load_dga("../data/dga-cryptolocke-1000.txt")
x_2,y_2=get_uniq_char_num(x2_domain_list)
x3_domain_list = load_dga("../data/dga-post-tovar-goz-1000.txt")
x_3,y_3=get_uniq_char_num(x3_domain_list)
```

以域名长度为横轴，唯一字母数字比例为纵轴作图：

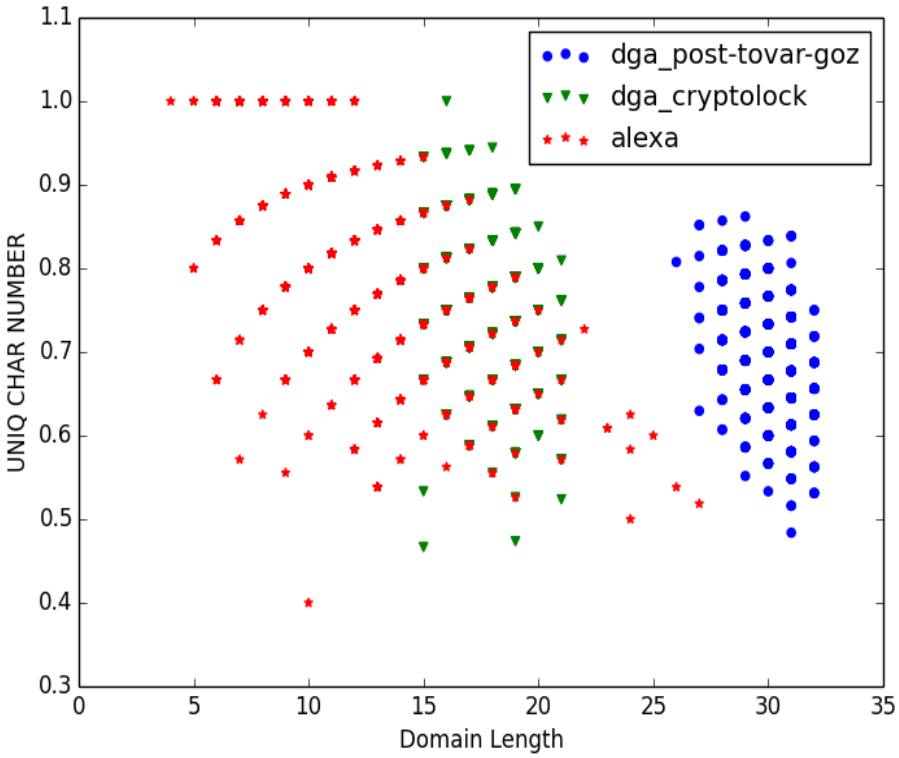

图9-15　　唯一字母数字比例分布图

分析图表，不同家族之间具有明显聚合效果，正常域名与 DGA 之间具有一定的区分性。

3　平均 jarccard 系数

jarccard 系数的定义为两个集合交集与并集元素个数的比值，本例的 jarccard 系数是基于 2-gram 计算的。

计算两个域名之间的 jarccard 系数的方法为：

```
def count2string_jarccard_index(a,b):
    x=set(' '+a[0])
    y=set(' '+b[0])
    for i in range(0,len(a)-1):
        x.add(a[i]+a[i+1])
    x.add(a[len(a)-1]+' ')
```

```
    for i in range(0,len(b)-1):
        y.add(b[i]+b[i+1])
    y.add(b[len(b)-1]+' ')
    return (0.0+len(x-y))/len(x|y)
```

计算两个域名集合的平均 jarccard 系数的方法为：

```
def get_jarccard_index(a_list,b_list):
    x=[]
    y=[]
    for a in a_list:
        j=0.0
        for b in b_list:
            j+=count2string_jarccard_index(a,b)
        x.append(len(a))
        y.append(j/len(b_list))
    return x,y
```

分别获取两个僵尸网络 DGA 域名数据以及 alexa 域名数据并计算元音字母比例：

```
x1_domain_list = load_alexa("../data/top-1000.csv")
x_1,y_1=get_jarccard_index(x1_domain_list,x1_domain_list)
x2_domain_list = load_dga("../data/dga-cryptolocke-1000.txt")
x_2,y_2=get_jarccard_index(x2_domain_list,x1_domain_list)
x3_domain_list = load_dga("../data/dga-post-tovar-goz-1000.txt")
x_3,y_3=get_jarccard_index(x3_domain_list,x1_domain_list)
```

以域名长度为横轴，平均 jarccard 系数为纵轴作图：

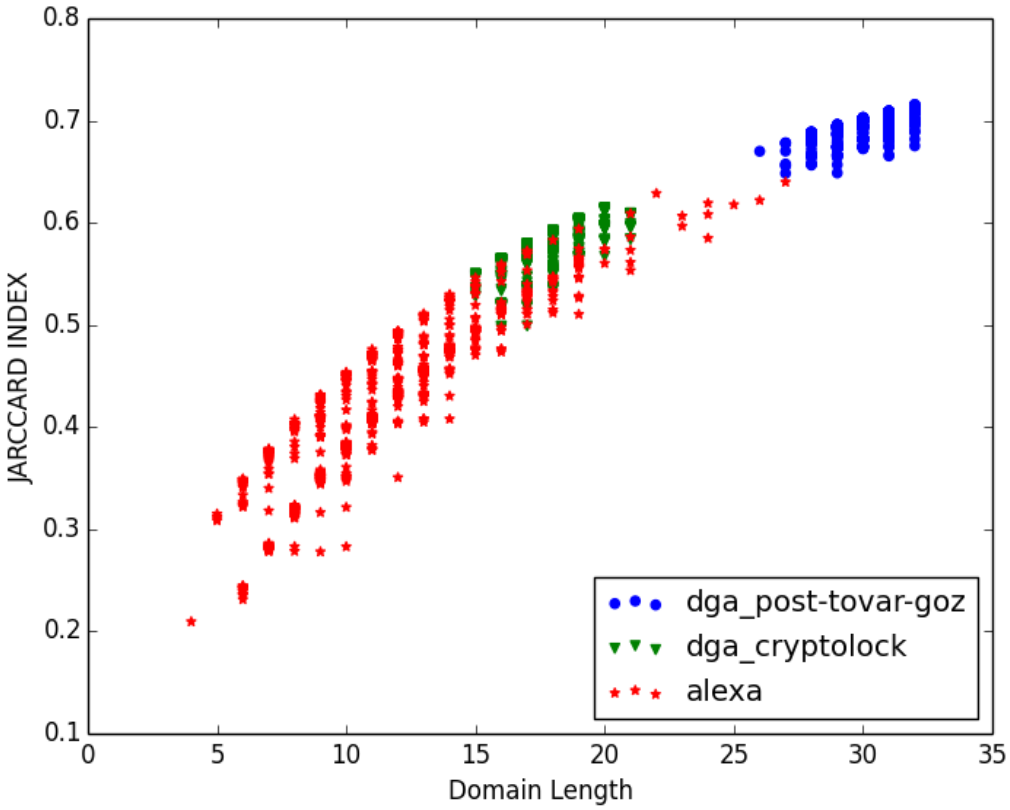

图9-1 平均 jarccard 系数分布图

分析图表，不同家族之间具有明显聚合效果，正常域名与 DGA 之间具有一定的区分性。

4 HMM 系数

正常人通常在取域名的时候，都会偏向选取常见的几个单词组合，抽象成数学可以理解的语言，就是以常见英文单词训练 HMM 模型，正常域名的 HMM 系数偏高，僵尸网络 DGA 域名由于是随机生成，所以 HMM 系数偏低。由于 HMM 将会在后面章节专门介绍，所以这里只是以统计图介绍下。

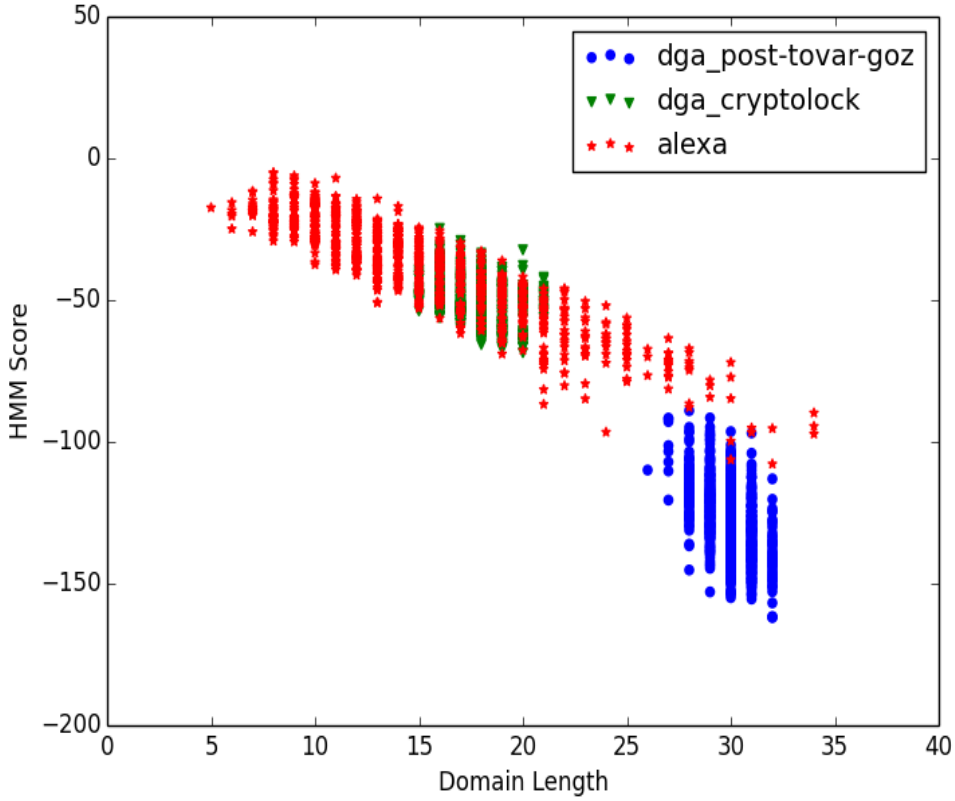

图9-2 HMM 系数分布图

9.5.3 模型验证

得到特征化的向量后使用 SVM 算法即可，这里就不赘述了。

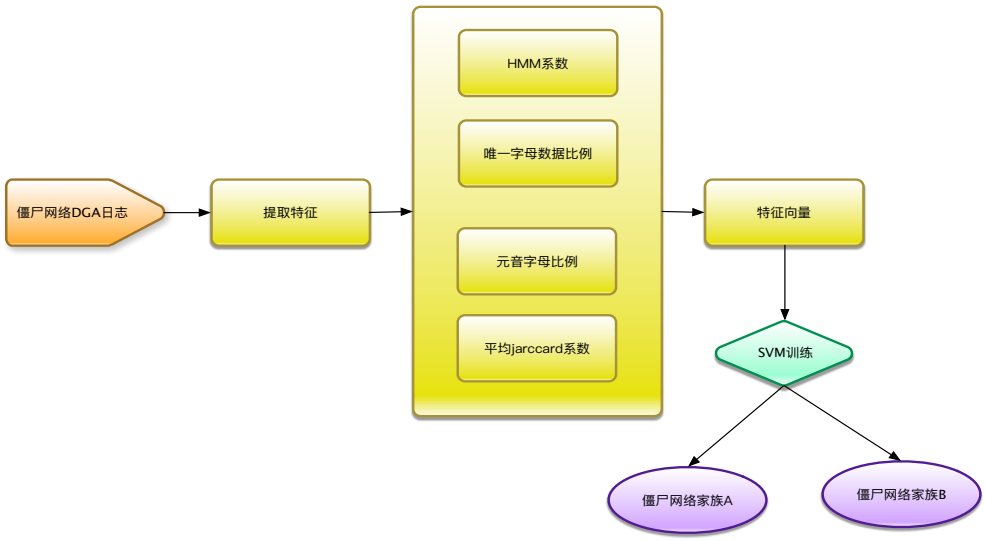

图9-3 SVM 验证 DGA 示意图

9.6 本章小结

本章介绍了支持向量基的基本原理，并以 XSS 检测以及 DGA 家族区分为例，实战了 SVM 的基本用法。支持向量机作为机器学习算法届的网红，一直在生活中的众多领域发挥作用。

9.6.1 参考文献

1) http://baike.baidu.com/
2) https://www.reddit.com/r/MachineLearning/comments/15zrpp/please_explain_support_vector_machines_svm_like_i/
3) http://blog.csdn.NET/lvhao92/article/details/50817110
4) http://Scikit-Learn.org/stable/
5) http://blog.csdn.net/v_july_v/article/details/7624837
6) Antonakakis, M., Perdisci, R., Dagon, D., Lee, W., Feamster, N.: Building a dynamic reputation system for dns. In: USENIX Security
7) Antonakakis, M., Perdisci, R., Lee, W., Vasiloglou, N., Dagon, D.: Detecting malware domains at the upper DNS hierarchy. In: USENIX Security
8) Antonakakis, M., Perdisci, R., Nadji, Y., Vasiloglou, N., Abu-Nimeh, S., Lee, W., Dagon, D.: From throw-away traffic to bots: detecting the rise of DGA-based malware. In: USENIX Security, USENIX Association
9) Bilge, L., Balzarotti, D., Robertson, W., Kirda, E., Kruegel, C.: Disclosure: detecting botnet command and control servers through large-scale netflow analysis. In: ACSAC. ACM
10) Bilge, L., Kirda, E., Kruegel, C., Balduzzi, M.: Exposure: Finding malicious domains using passive DNS analysis. In: NDSS
11) Han, J., Kamber, M.: Data mining: concepts and techniques. Morgan Kaufmann
12) Holz, T., Gorecki, C., Rieck, K., Freiling, F.C.: Measuring and detecting fast-flux service networks. In: NDSS
13) Leder, F., Werner, T.: Know your enemy: Containing conficker. The Honeynet Project, University of Bonn, Germany, Tech. Rep.
14) http://www.freebuf.com/articles/network/114693.html

15) J Choi，H Kim，C Choi，P Kim,Efficient Malicious Code Detection Using N-Gram Analysis and SVM

16) J Hong，S Kim，J Park，J Choi,A Malicious Comments Detection Technique on the Internet using Sentiment Analysis and SVM

17) G. Buehere, B.W. Weide and P.A. Sivilotti, Using Parse Validation to Prevent SQL Injection Attacks, InProceedings of the 5th international Workshop on software Engineering and Middleware, 2005,

18) B.-Y. Zhang, J.-P. Yin, J.-B. Hao, D.-X. Zhang and S.-L. Wang, Using SupportVector Machine to Detect Unknown Computer Viruses, International Journal ofComputational Intelligence Research，2006

19) B. Zhang, J. Yin and J. Hao, Intelligent Detection Computer Viruses Based onMultiple Classifiers, Ubiquitous Intelligence and Computing，2007

20) C. Fung, Collaborative Intrusion Detection Networks and Insider Attacks, Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications(JoWUA)，2011