背景

## 基于LSB的图像隐写技术

### 隐写术

隐写是指把一个文件、消息、图像或者视频隐藏到另一个文件、消息、图像或者视频的行为。与密码学不同的是，隐写术旨在隐藏消息或其他形式的信息本身的存在，不引起发送方和接收方以外的人的怀疑而完成信息的交流，而密码学则用于隐藏这些信息的内容，使得非发送方或接收方即使截获消息也无法得到所交流的信息的真实内容。隐写术的主要任务是使发生在公共信道上的秘密信息交流不被察觉，隐藏了秘密信息的图片或其他格式的载体与隐藏之前的原始数据在视觉上以及其他几个重要特征一致。

隐写术是一种秘密通信的艺术，这个术语最早在从几千年前开始，人类就着迷于密文书写，并出于多种原因和动机学习这种和研究这种技术[1]，早期密写通常是指密码学，而随着时代发展现在密写也涵括了隐写术，两者用不同的方式实现密写的目的。无论在哪个时代，隐写术广泛应用于各个领域。早期的隐写实践是使用不可见的墨水在信件中书写消息，显而易见地，军事和政治中，在不被敌方察觉的前提下向友方传递信息的能力十分关键。在数字时代，这种思想发展为在多媒体文件中隐藏其他数字消息。现代隐写术在1985年以后随着个人计算机的推广而问世，随着计算机科学技术和数学的发展以及深入研究而迅速发展并投入更广泛的应用，例如电子通信包含了传输层内的隐写码以传送如文档文件或图片文件等多媒体数据。

与密码学相似，隐写术的使用场景决定了它必须满足以下三个要求：

* 保密性：不容易被探查到隐藏消息的存在
* 可获得性：不会出现由于修改数据的载体导致秘密消息的丢失，秘密消息可以被恢复
* 完整性： 其他人无法伪造出错误信息

隐写系统可以被视为加密系统的一个特例[2]，在这个系统中我们要求密文与明文对于其他人来说难以区分。值得注意的是，隐写加密方必须首先合成一个与秘密消息无关的无害的文件。载体合成具有很大的挑战性，高效合成载体的隐写机制很少。对于这个问题，使用通过修改实现隐写的方法很好地避免了这个挑战。这种系统使用一个现有的与秘密消息无关的文件，也就是载体，作为隐写系统的输入的一部分，接下来秘密消息转换为以让人难以觉察的载体的修改，得到与载体极其相似的结果，即伪装（stego）。理论上，载体合成是最自由且强大的隐写方法，因为它可以不受限制地适用于任意的秘密消息。但在实践中，无害文件的合成是一个非常复杂且低效的过程，所以大多数众所周知的实用系统实用的都是修改载体的方法。当然除了合成和修改两种模式以外，载体选择[3]也是可行的方案，可以类比为传统密码学中使用的编码本，有多个载体或密文分别对应特定的秘密消息。然而，这种模式需要的可用载体数量太大，并不是在所有场景下都实用。所以我们在大多数情况下会选择通过修改进行隐写的模式，也就是说隐写的过程需要使用已经存在的文件作为原始的输入，而多媒体文件（如图像、音频和视频等）往往较大，包含了大规模的数据，可以找到足够的空间隐藏消息同时在不表达出可以被察觉的异常效果，是理想的载体。其中，数字图像的应用场景广泛，修改方便，且容易在互联网快速传播，成为了应用最多的载体。本文也将围绕图像隐写技术展开。

早期的隐写安全完全依赖于隐写算法，只将载体和秘密消息作为输入而不使用秘钥，被称为纯隐写系统，一旦隐写算法泄露则整个系统被破解。将秘钥引入作为输入的隐写系统则被称为秘钥隐写系统。在密码学中，加密者和解密者共享秘钥，关于算法的知识对双方区分伪装和正常消息没有帮助，这个结论即kerckhoffs原则，在隐写系统中并不总是成立[4]。符合kerckhoffs原则的系统在使用一个秘钥的实例被攻破后，使用其他秘钥的实例的安全性不受影响，与其他系统相比这样的系统具有巨大的优势，因为生成不同的秘钥恨容易，而重新设计一个算法却很困难。但事实上，基于kerckhoffs原则安全的隐写系统很难设计，同时大多数应用场景下，除了发送方和接收方外其他人对于他们的隐写系统通常一无所知，因此，不遵循kerckhoffs原则的系统也是可行的。

### LSB隐写算法

对于一个二进制整数来说，最低有效位（LSB）是最低的比特位（即第0位），决定了这个数是奇数还是偶数，这个比特位相比于其他位置的变化对于整个数值变化影响是最小的。

LSB嵌入方法是一种经典的图像隐写算法。这种方法最早被用于像素图像，在像素图像中，每个像素都是代表该点颜色强度的整数。在灰度图像中，每个点的像素值表达了该点色彩介于黑白之间的程度，而在具有三个色彩信道的RGB图像中，每个像素点由三个独立个代表红、绿、蓝三种颜色强度的值合成单元，这些像素值的取值范围通常是8bit的整数值，也就是。在颜色强度上的微小改动被察觉的可能性很小，LSB隐写算法正是利用了像素图像的这个特性，舍弃每个像素原来的最低有效位（LSB），并替换为需要隐藏的消息。接收者得到伪装完成的图像后可以通过模2操作提取新的LSB并将之还原为完整的消息。

最基础的基于LSB的隐写系统从图像的左上角开始逐位嵌入秘密消息，推进的顺序是由发送者和编程语言决定的最自然的方向。为了方便接受者确定隐藏的消息在何处结束，我们可以将消息的大小作为头部隐藏在前像素中，这里的是一个双方已经约定好的整数。通过这个简单的python程序可以演示这个隐写的过程（在python中最自然的方向是逐行嵌入）：

def convert\_message\_to\_bit(string\_text):  
 '''convert the secret message from string to bytearray'''  
 bit\_string=[]  
 for char in string\_text:  
 bit\_string.extend([int(d) for d in bin(ord(char))[2:].zfill(8)])  
 return np.array(bit\_string)  
  
def hide\_side\_information(carrier,length,pro\_text\_size):  
 '''hide some side information(i.e. the length of secret message here) in the front of the image'''  
 bit=bin(length)[2:].zfill(pro\_text\_size)  
 length\_data=np.array([int(d) for d in bit ])  
 side\_info=carrier[:pro\_text\_size]+length\_data  
 carrier[:pro\_text\_size]=side\_info  
 return carrier  
  
def binary\_array\_to\_int(arr):  
 '''convert a 0-1 array into a decimal integer'''  
 bit\_string=''.join(arr.astype(np.str))  
 return int(bit\_string,2)  
  
def hide(cover\_pic,secret\_text,pro\_text\_size):  
 '''hide secret information in a given picture(cover), return a image object'''  
 length=len(secret\_text)  
 cover=np.array(cover\_pic)  
 size=cover.shape  
 cover\_data=cover.flatten()  
 carrier\_data=cover\_data & 0b11111110 #truncated cover without LSB plane  
 l=hide\_side\_information(carrier\_data,length,pro\_text\_size)  
 secret\_bytes=convert\_message\_to\_bit(secret\_text)  
 secretbytes\_length=length\*8  
 carrier\_data[pro\_text\_size:pro\_text\_size+secretbytes\_length]=carrier\_data[pro\_text\_size:pro\_text\_size+secretbytes\_length]+secret\_bytes  
 new\_data=carrier\_data.reshape(size)  
 new\_iamge=Image.fromarray(new\_data)  
 return new\_iamge  
  
def reveal(secret\_iamge,pro\_text\_size):  
 '''extract secret message from a given picture(stego) return a secret string'''  
 im=np.array(secret\_iamge).flatten()  
 lsb\_plane=im & 0b00000001  
 length=binary\_array\_to\_int(lsb\_plane[0:pro\_text\_size])  
 secret\_bytes=lsb\_plane[pro\_text\_size:pro\_text\_size+length\*8].reshape((length,8))  
 secret\_bit=[binary\_array\_to\_int(x) for x in secret\_bytes]  
 return ''.join(chr(c) for c in secret\_bit)

这种方法使得隐藏的消息总是在同样的位置。显然，作为最经典的图像隐写算法，LSB嵌入模式许多的变体，针对这个问题，如果随机选择用于嵌入消息的像素点的位置则可以让伪装变得更加隐蔽。在伪装系统中引入伪随机数生成器（PRNG），依据其产生的序列选择嵌入消息的像素位置，可以使伪装消息的分布更加随机。这个伪随机数生成器的种子作为系统的秘钥必须在之前就由双方完成交换。

除此之外，还有一种经典的基于LSB嵌入的改进，被称作LSB匹配，体现在随机化了每个样本的篡改。不同于经典LSB嵌入直接舍弃最低有效位的操作，LSB匹配将观察该像素的最低有效位与需要嵌入的信息单元的关系，如果一致则保持最低有效位不变，否则将等概论地对该像素值，具体来说对于偶数值像素而对于奇数值像素。消息的提取同样可以使用模2运算直接完成。

基于LSB的多种隐写方法被广泛应用在空域（如像素图像）并推广到了其他整数信号发生幅度的变化难以被觉察到的场景。对于LSB隐写算法进行在多个方面进行一些微小的改进，就可以获得在隐藏效果上得到巨大的提升，因此现有的工作往往针对于LSB具体实现参数的调整而非设计LSB以外的隐写算法。

## 隐写分析

隐写分析是指对于使用隐写术隐藏的秘密消息的探查。隐写分析的主要任务是识别给定的文件中是否隐藏了可疑的秘密消息，最基础的隐写分析算法以可疑文件作为输入，而将二元分类的“是”和“否”作为输出结果。在此基础之上，有些隐写分析也加入了恢复隐藏信息的功能，但是作为隐写分析最根本的任务是探查是否存在隐藏的消息，一旦探查到隐藏消息的存在，甚至不需要恢复出具体的消息，就可以认为该隐写系统被攻破。

图像隐写分析方法根据手段不同主要分为四种：视觉隐写分析，结构隐写分析，统计隐写分析和学习隐写分析。

视觉隐写分析是最容易的分析手段，即观察图像是否存在视觉上的异常痕迹，最常见的实现方式即提取图像的LSB平面。通过修改图像的LSB而达到隐藏效果的隐写图像相比于自然图像在LSB平面体现出一定的异常，特别是在与高位比特平面的图像对比时，不自然的人工痕迹更为明显。

结构隐写分析则寻找在媒体表达或文件格式上泄露的痕迹。由于人为限制，某个版本的隐藏工具会对图像的大小有所要求，比如Seek and Hide 4.1[5]只会使用像素的图像作为载体，那么我们可以根据这个特性重点探查像素的图像。再如JPEG兼容压缩攻击[6]：JPEG图像使用的是有损压缩，不是每个像素组都会成为可能的一个给定的JPEG解压实现的可能输出，因此这种攻击方法可以针对载体之前使用过JPEG压缩的像素图像的嵌入模式进行分析。

统计隐写分析借鉴统计学的方法探查隐写。这种方法通常需要一个统计模型来描述伪装和/或载体的概率分布，因此我们需要预先通过分析相关载体和隐写系统建立一个这样的数据模型。统计假设测验是最典型的统计方法，用以判定一个可疑的文件是可信的载体还是伪装。不仅如此，一些量化的分析使用了参数估计来估测嵌入消息的长度。对于载体的数据建模被证明是困难的，但是对于给定的隐写系统，却往往容易建立良好的关于伪装数据的模型，因此可以有针对地应用于探查某种隐写方法。

但在实际场景中很难单纯通过分析的方法建立一个统计模型，对此我们可以借助计算的方法，使用学习隐写分析，也就是使用穷举的思想得到经验模型。具体的解决方案可以使用模式识别、机器学习或人工智能等领域的方法。

针对基于LSB的图像隐写分析的方法主要有测验[7]，样本对分析[8]和RS隐写分析[9]，他们的主要特性如表所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 方法 | 原则 | 特征 | 作为探查分类器 |
| 测验 | 将图像进行分片并对每个分片使用测试 | 系数 | 基于阈值 |
| RS隐写分析 | 使用翻转和掩码来识别R,S,U群并画出RS图进行估计 | 在不同的掩码中R,S,U群出现的频率 | 基于阈值 |
| 样本对分析 | 计算样本对的频率，解方程估计 | 每个特定样本对出现的频率 | 基于阈值 |

### 测验

测验，也就是值对（PoV）攻击，是统计隐写分析的先驱。攻击的设计基于图像直方图来检测LSB嵌入。

为了理解值对，我们可以先观察载体和使用LSB满容量嵌入得到的伪装的直方图，直方图统计了每个灰度值出现的次数，可以从图中看出伪装图像的直方图中有更多的成对出现的大致等高的条方。

回顾LSB嵌入的过程，载体图像中一个值为的像素可能在依然保留值或者改变为，同理，一个值为的像素也可能保持不变或者变为，因此对于每一个的都形成了一个值对，LSB嵌入可以改变一个值对内的两个像素值但无法将一个像素值从一个值对改变到另一个值对。如果嵌入的消息是一个随机消息或者加密过的消息，也就是需要嵌入的位近似于0和1的均匀分布，那么对于伪装图像的每一个来说，和也是近似于均匀分布的。

基于上述特性，我们可以使用统计假设测验来检查可疑的图像是否为包含嵌入消息的伪装图像。这里选择使用测验来检验像素值和出现的频率是否符合均匀分布。令表示颜色强度值出现的次数，在（满容量）伪装图像中我们期望一个值对中的两个值出现的频率符合均匀分布，因此对所有有，根据标准测试[10]有

查询分布表我们可以计算对应的值，用以度量图像为伪装图像的概率，根据值大小决定该图像是否包含隐藏消息。

这样的测验一开始只适用于探查是否存在连续的LSB替换，对于低嵌入率的情况效果不佳，且对于使用了伪随机数决定嵌入像素的隐写系统无效。随着现有研究的发展，测验的思想被推广，可以通过多次渐进迭代、从特定位置开始探查、增大样本大小等方法[11]改进用于随机位置的LSB嵌入以及其他LSB衍生系统的探查。

### 样本对分析

样本对分析（SPA）方法跟踪LSB嵌入前后的样本对构成的多个集合，并分析多个集合间的关系计算嵌入消息的长度。

假设图像用连续的样本表示，其中为划分的样本总数，一个样本对，，集合为图像中所有样本构成的集合，我们定义是包含类似于或的的子集，其中。对于，定义为前7比特的差值的样本对。我们再定义以及，对于且有。如果中的样本对离散均匀分布，那么对于任意的满足的都有

这也是SPA方法的关键理论基础。

显然集合中的样本对都形如或，中的样本对则形如或，如果我们考虑LSB嵌入中样本对的翻转，我们有四种修改模式：00,01,10,11，其中1表示样本对中LSB发生改变的样本，而0表示保持完好的样本，所以对于每一个，多元子集都被划分为，和，那么显而易见，这些在嵌入后都是相近的，而，和却不是。所以我们继续把分割为和，这时包含形如值为或的样本对，包含形如包含形如值为或的样本对，的多重集合通过上述操作可以被分为，，和，成为的变多重集合，那么在LSB嵌入后，我们可以获得一个描述变多重集合间转换的有限状态机。

根据[8]中提到的方法我们可以用如下的方程计算嵌入消息的长度:

我们令即假设只使用最低一位进行嵌入，则可以估计嵌入消息长度的最小值。

### RS隐写分析

RS方法挖掘伪装图像空间的相关性。相比于用于嵌入操作的原始载体，LSB嵌入后得到的伪装图像的空间相关性往往有所下降，我们可以利用这个特点探查隐写的存在。RS方法的主要思想如下：先将图像划分为大小相等的小图像块（群)，对得到的每个小图像块都分别进行非负翻转和非正翻转操作，再对比统计经过这些操作以后小图像块的空间相关性的变化，Fridrich等人经过统计分析认为，LSB嵌入前后空间相关性增加的小图像块数量和相关性减小的小图像块数量各自呈现一定的关系，可以根据这样的关系判断图像中是否有经过LSB嵌入的隐藏消息。

我们规定三种翻转操作：正翻转，负翻转和零翻转。对于任意的，正翻转是像素值在和间的转换，如,,,，是使用LSB嵌入秘密消息过程中发生的翻转操作。负翻转则是像素值在和间的转换，如,,,。零翻转则保持原像素值不变。原始的LSB嵌入过程可以描述为如下过程：当嵌入的位与像素的LSB位相同时对该像素进行零翻转，否则使用正翻转。

我们定义作用在群的函数为平滑度函数，表示为，图像的平滑度函数函数是每个像素与其邻接像素的绝对差值的和，反映了图像的空间相关性。平滑函数的值越大，图像的混乱程度越大，图像空间的相关性越小。

我们对于每个组都使用掩码为的翻转操作，得到的翻转结果为其中。根据翻转前后像素组值大小的变化我们可以定义三类像素组,和：如果则为正则组，记为；如果则为奇异组，记为；除此以外，若则为不变组，记为。 掩码表示的非负翻转操作下的正则组的相对数量记为，奇异组的相对数量记作，同理对于掩码为的情况，即非正翻转操作得到的正则则的相对数量记为，奇异组的相对数量表示为。

Fridrich等人提出零假设是对于典型的未经过LSB嵌入的载体图像满足和的大小近似相等，同样和也是近似相等的关系，也就是经过非负翻转后空间相关性减小的小图像块和经过非正翻转后空间相关性减小的小图像块数量大致相等，并且对于相关性增大的小图像块这种类似的关系也成立。同时，对于未经LSB嵌入的图像还有和，这意味着对于自然图像，非零翻转必然使图像空间的像素相关性呈现下降趋势。对于经过LSB嵌入消息的伪装图像，应用非负翻转后的空间相关性相较于应用非正翻转有很大提升，所以有和。

由于LSB嵌入本质上应用的也是一种非负翻转的实例，所以在统计意义上，随着嵌入率的上升，和的差值会有所减小。然而在对伪装图片进行非正翻转的过程中，有些像素可能经历过正负两次翻转，致使其与原始值偏离得更远，因此和不会随着嵌入率的上升而发生显著变化。根据和的差值可以量化估计嵌入消息的长度，但是由于计算量较大这里不作介绍，RS隐写分析的主要实现是捕获两类翻转后图像空间相关性的非对称性变化的特点发现图像中LSB隐写痕迹。

## 支持向量机

通过以上对隐写分析方法的介绍，容易发现单纯的LSB替换或者LSB匹配等基于LSB嵌入的隐写方案在安全性方面存在一定的缺陷，易受到这些隐写分析方法的攻击。针对这个问题，我们可以引入学习的概念，寻找适合隐藏密码消息的像素的位置，相比按照顺序连续嵌入或使用伪随机数生成器寻找嵌入位置的隐写方法，通过学习现有样本和及对应的评估结果的关系，相比于传统的对抗隐写分析的方法可以省略复杂的分析过程，基于经验得到直观的选择嵌入位置的方法。这里的过程可以被简述为生成一些在不同图像上选择不同位置的LSB嵌入的伪装图像，将它们的隐写分析结果连同位置和图像的几个特征作为训练集，使用支持向量机（SVM）进行学习，完成有监督学习得到的分类器用于对需要嵌入消息的图像的位置（像素）进行二元分类，输出结果为适合隐藏消息的位置的集合和不适合隐藏消息的位置的集合。

能实现分类效果的机器学习模型算法数不胜数，每种算法都存在各自独有的优势和劣势。本文选择SVM是出于其能最大限度区分出每个像素点是否适合隐藏秘密数据的考虑，不仅如此，SVM理论和实践发展较为成熟，关于SVM本身的优化和参数调节的研究丰富，易于实现，在不同应用领域的适用性都比较理想，适合用于支持LSB隐写方法的改进优化。

对于一个给定的容量为的训练样本集，其中，如果存在可以根据的值线性划分整个样本空间的超平面，那这样的超平面往往不止一个，但是需要挑选最佳的超平面作为分类器。 SVM方法选择这个最佳的超平面的标准是使两类样本向量中距离超平面最近的向量到超平面的距离之和最小，这个距离之和也被称为间隔，同时，这些到每个类的样本中到超平面距离最近的向量则被称为是支持向量。

SVM的输出结果是超平面的法向量和样本空间中的截距，这两个系数可以唯一确定一个可用于样本分类的超平面，对于任意的向量，代入函数可以得到分类结果。

SVM的训练过程是找到这样一个可以分割不同类样本的超平面并使其中的间隔最大，可以表示为优化过程：

但是直接从此式入手求解过程有些复杂，使用Lagrange乘子法可以把这个问题转化为对应的对偶形式，即凸二次规划问题：

求解得到维的Lagrange乘子向量,系数，根据KKT条件中的约束，找到一个对应的Lagrange乘子为0的向量，也就是支持向量，代入求得的值。这样简化后的过程可以通过简单的编程完成，也可以使用SMO算法[12]等技巧提升计算速度。

原始的SVM方法如上所述，可以解决大部分的简单分类问题，但是依然存在一定的局限，如对于线性不可分的训练集无法得到闭解，以及对于最大间隔的追求可能导致出现过度拟合。所以在基本的SVM之上可以引入核技巧或者软间隔的概念提升SVM的适用性。

对于线性不可分的情况，我们可以将维向量变换为新的维特征空间的向量，一般来说低维无法解决的问题往往向高维寻求解决方法，因此，通常有。如果我们规定新的特征空间中两个向量的内积可以用函数来表示，即，这个函数被称为核函数。使用核函数代换原来的对偶问题中目标函数的低维内积操作为，以及最后的训练结果也就是分类函数

，这种计算技巧被称为是核技巧，在原特征向量空间无法解决或者解决效果不理解的SVM问题中，应用特征空间变换操作可以把样本映射到高维的特征空间寻找可行的解，但是由于我们很难直接计算高维内积，所有可以直接使用现有的核函数计算内积，也等价于应用了该核函数对应的空间变换操作。在设置SVM参数的过程中，选择核函数即选择空间变换的具体形式，如应用可以把原向量映射到无限维空间的高斯核（下文提到RBF核为其中的一种特例）就相当于选择训练和预测过程中把样本变换为无限维。所以，选择一个合适的核函数对于训练效果和预测性能是至关重要的，以下为常见的核函数：

|  |  |  |
| --- | --- | --- |
| 核函数 | 表达式 | 参数 |
| 线性核 |  | 无 |
| 多项式核 |  | 为多项式的最高次数，为系数，为常数项 |
| RBF核 |  | 系数 |
| sigmoid核 |  | 系数，常数项 |

除此之外，经过证明，当且仅当上对称函数对于任意的维向量组成的数据集的运算得到的维核矩阵是半正定矩阵时，这个函数有对应的空间变换映射的，可以作为SVM的核函数使用[13]。因此，除了这些常用的核函数，在实际应用中也可以通过对常用核函数进行线性组合、求直积和其他方法设计实用的核函数。

[1] SCHAATHUN H G. Machine learning in image steganalysis[M]. Chichester West Sussex UK: Wiley-IEEE Press, 2012.

[2] BACKES M, CACHIN C. Public-key steganography with active attacks[G]//Theory of cryptography. Springer, 2005: 210–226.

[3] FILLER T, FRIDRICH J. Complete characterization of perfectly secure stego-systems with mutually independent embedding operation[C]//Acoustics, speech and signal processing, 2009. iCASSP 2009. iEEE international conference on. IEEE, 2009: 1429–1432.

[4] FRIDRICH J, GOLJAN M, SOUKAL D. Searching for the stego-key[C]//Electronic imaging 2004. International Society for Optics; Photonics, 2004: 70–82.

[5] PETER W. Disappearing cryptography[J]. Massachusetts, AP Professional, 1996.

[6] FRIDRICH J, GOLJAN M, DU R. Steganalysis based on jPEG compatibility[C]//ITCom 2001: International symposium on the convergence of iT and communications. International Society for Optics; Photonics, 2001: 275–280.

[7] WESTFELD A, PFITZMANN A. Attacks on steganographic systems[C]//Information hiding. Springer, 1999: 61–76.

[8] DUMITRESCU S, WU X, MEMON N. On steganalysis of random lSB embedding in continuous-tone images[C]//Image processing. 2002. proceedings. 2002 international conference on. IEEE, 2002, 3: 641–644.

[9] FRIDRICH J, GOLJAN M, DU R. Reliable detection of lSB steganography in color and grayscale images[C]//Proceedings of the 2001 workshop on multimedia and security: New challenges. ACM, 2001: 27–30.

[10] JOHNSON R, BHATTACHARYYA G. Statistical concepts and methods[J]. Wiley series in probality and mathematical statistics., 1977.

[11] PROVOS N, HONEYMAN P. Hide and seek: An introduction to steganography[J]. Security & Privacy, IEEE, IEEE, 2003, 1(3): 32–44.

[12] PLATT J, OTHERS. Sequential minimal optimization: A fast algorithm for training support vector machines[J]. technical report msr-tr-98-14, Microsoft Research, 1998.

[13] SCHÖLKOPF B, SMOLA A J. Learning with kernels: Support vector machines, regularization, optimization, and beyond[M]. MIT press, 2002.