entropy

MDPI

Article

# Kernel Mixture Correntropy Conjugate Gradient Algorithm for Time Series Prediction

Nan Xue [1,2,3], Xiong Luo [1,2,3,*], Yang Gao [4], Weiping Wang [1,2,3], Long Wang [1,2,3], Chao Huang [1,2,3] and Wenbing Zhao [5]

1 School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China
2 Institute of Artificial Intelligence, University of Science and Technology Beijing, Beijing 100083, China
3 Beijing Key Laboratory of Knowledge Engineering for Materials Science, Beijing 100083, China
4 China Information Technology Security Evaluation Center, Beijing 100085, China
5 Department of Electrical Engineering and Computer Science, Cleveland State University, Cleveland, OH 44115, USA
* Correspondence: xluo@ustb.edu.cn; Tel.: +86-10-6233-2526

Received: 17 June 2019; Accepted: 9 August 2019; Published: 11 August 2019

**Abstract:** Kernel adaptive filtering (KAF) is an effective nonlinear learning algorithm, which has been widely used in time series prediction. The traditional KAF is based on the stochastic gradient descent (SGD) method, which has slow convergence speed and low filtering accuracy. Hence, a kernel conjugate gradient (KCG) algorithm has been proposed with low computational complexity, while achieving comparable performance to some KAF algorithms, e.g., the kernel recursive least squares (KRLS). However, the robust learning performance is unsatisfactory, when using KCG. Meanwhile, correntropy as a local similarity measure defined in kernel space, can address large outliers in robust signal processing. On the basis of correntropy, the mixture correntropy is developed, which uses the mixture of two Gaussian functions as a kernel function to further improve the learning performance. Accordingly, this article proposes a novel KCG algorithm, named the kernel mixture correntropy conjugate gradient (KMCCG), with the help of the mixture correntropy criterion (MCC). The proposed algorithm has less computational complexity and can achieve better performance in non-Gaussian noise environments. To further control the growing radial basis function (RBF) network in this algorithm, we also use a simple sparsification criterion based on the angle between elements in the reproducing kernel Hilbert space (RKHS). The prediction simulation results on a synthetic chaotic time series and a real benchmark dataset show that the proposed algorithm can achieve better computational performance. In addition, the proposed algorithm is also successfully applied to the practical tasks of malware prediction in the field of malware analysis. The results demonstrate that our proposed algorithm not only has a short training time, but also can achieve high prediction accuracy.

**Keywords:** kernel adaptive filtering; conjugate gradient; correntropy; sparsification criterion; malware prediction

## 1. Introduction

Usually, traditional time series prediction methods mainly include autoregression, Kalman filtering, and moving average models. These traditional approaches focus on mathematical statistics and have no capabilities of self-learning, self-organization, and self-adaption. In particular, they cannot be effectively used for data types of nonlinear and multi-feature dimensions in analyzing some complex problems. Currently, there are some machine learning methods developed to address this issue, such as support vector machine (SVM), artificial neural network (ANN), and deep neural network (DNN), then some satisfactory results are achieved in dealing with practical applications, e.g., malware analysis [1,2]. However, there still exists several issues that need to be addressed in using SVM, ANN, and DNN, such as long training time and difficulty in parameter determination. Therefore, it has become critical to seek a better machine learning model [3].

The kernel method as an effective nonlinear system modeling technique has been widely used in the machine learning community [4,5]. Among the available kernel learning methods, the kernel adaptive filtering (KAF) has become a popular calculation method with good computing performance, which has been successfully employed in signal processing, time series prediction, and many others. The main idea of KAF is to map the input data in the original space to the high-dimensional feature space, that is the reproducing kernel Hilbert space (RKHS) [6]. Then, a linear algorithm in this high-dimensional feature space can be applied to solve the nonlinear problem in the original space. The disadvantage of the feature space is that the computational complexity of algorithm increases exponentially with the dimensionality. However, because the kernel method is implicit, it uses kernel function calculation to replace the inner product calculation in high-dimensional feature space. Thus, the issue that the computational complexity increases rapidly with the dimensionality could be effectively avoided.

With the comprehensive study of nonlinear adaptive filters, more extended KAF algorithms have been developed in recent years. In this case, some popular KAF algorithms include kernel least mean squares (KLMS) [7] and kernel recursive least squares (KRLS) [8]. The algorithm KLMS has been widely used in the field of adaptive signal processing due to its simplicity and efficiency [9]. The algorithm KRLS is designed by recursively solving the least squares (LS) problem. Compared with KLMS, KRLS achieves better filtering accuracy and faster convergence speed at the cost of increased computing and storage. Meanwhile, the conjugate gradient (CG) method is another optimization strategy, and it provides a tradeoff between convergence rate and computational complexity, which can generate a better optimal solution than the stochastic gradient descent (SGD) method [10]. Therefore, the method CG has been successfully applied to KAF, and the kernel conjugate gradient (KCG) algorithm was proposed [11]. The KCG with low computational complexity can achieve the same performance as KRLS.

However, the above algorithms are all deduced and experimented in a Gaussian noise environment. This assumption may not be accurate, since the system noise does not always follow the Gaussian distribution in practical applications. Actually, it is accompanied by some impulse noise with low frequency and large amplitude. The interference of this impulse noise on the system is not negligible. In this case, the KAF algorithm derived on the basis of Gaussian noise environment is sensitive to non-Gaussian noise or outliers. As a robust nonlinear similarity measure in kernel space, correntropy has received more and more attention in the field of machine learning and signal processing [12]. Correntropy can capture higher-order statistics of errors and provide significant performance improvement on filtering precision, particularly in non-Gaussian environments. Therefore, more and more algorithms use correntropy as a cost function to improve the stability of algorithms in a non-Gaussian noise environment [13,14]. Furthermore, on the basis of correntropy, the mixture correntropy was proposed [15]. It uses a mixture of two Gaussian functions as a kernel function to enhance flexibility and improve the performance.

Motivated by the work mentioned above, in order to improve the filtering accuracy, convergence speed, and robustness against impulse noise at the same time, the mixture correntropy criterion (MCC) [11] is applied to KCG method. Then, a novel kernel learning algorithm, called kernel mixture correntropy conjugate gradient (KMCCG), is accordingly proposed in this article. In fact, MCC cannot be directly applied to the CG method due to its nonconvexity [15]. Therefore, in accordance with the conjugate function theory, we use the half-quadratic optimization method to transform the mixture correntropy loss into the mixture correntropy half-quadratic function. Then, the mixture correntropy can be smoothly applied to the CG method. Finally, according to the kernel technique, the algorithm KMCCG is proposed for robust online kernel learning. Moreover, in the weight updating, the KAF embeds a growing memory structure, i.e., a growing radial basis function (RBF) network, which leads to the increase of the memory requirement and computation of KAF. There are many traditional methods to restrain the growth of the network structure, such as the novelty criterion [16], the correlation criterion [17], the approximate linear dependence criterion [8], and the surprise criterion [18]. Since the angle between two elements in the Hilbert space can be expressed by the inner product and the similarity of elements in Hilbert space can be measured by the angle, here we use a sparsification criterion based on the angle among elements. The angle criterion used here not only provides geometric intuition, but also offers a simple structure that can be implemented easily.

With the rapid advancement of Internet technology [19], the issue of network security imposes huge challenges to the Internet. Specifically, the demand for malware analysis has become increasingly urgent, and practitioners and researchers have been making progress in the field of malware prediction and detection [20]. Usually, malware is able to implement intention by calling the existing application programming interface (API) in the system. Therefore, the API calling time series as a software behavior is analyzed to achieve malware prediction and detection [21]. Generally speaking, the obtained API call time series can be used to predict future malicious behavior. Specifically, in this article, the proposed algorithm KMCCG is also used in the practical application of malware prediction.

The main contributions of this article are summarized as follows. (1) On the basis of mixture correntropy, a novel robust algorithm KMCCG is proposed through a comprehensive use of the half-quadratic optimization method, the CG technique, and the kernel trick. KMCCG cannot only improve the learning accuracy, but also maintain robustness to impulse noise. (2) In view of the issue that the algorithm KMCCG will produce a growing RBF network, the sparsification criterion based on the angle is used to control the network structure. (3) For a special time series analysis application in relation to malware prediction, KMCCG is accordingly used to achieve this task, which verifies that our proposed algorithm can achieve higher prediction accuracy with less training time.

The rest of this article is organized as follows. In Section 2, the mixture correntropy, the algorithm KCG, and the sparsification criterion are introduced. In Section 3, the details of our algorithm KMCCG are presented. In Section 4, the simulation on time series prediction and the experiment on the malware prediction task are conducted to verify the effectiveness of the our proposed algorithm. The conclusion is summarized in Section 5.

## 2. Related Work

### *2.1. Mixture Correntropy*

Correntropy is a local similarity function, which is defined as the generalized correlation in kernel space. It is closely related to the cross-information potential (CIP) in information theory learning (ITL) [22]. It shows very promising results in nonlinear non-Gaussian signal processing. The main property of correntropy is that it provides an effective mechanism to mitigate the influence of large outliers. Recently, correntropy has been successfully applied in various areas, such as signal processing [23], machine learning [24–26], adaptive filtering [27–29], and others [30–32].

The correntropy is used to represent the similarity between two random variables $X$ and $Y$. Let $k_\sigma(\cdot,\cdot)$ be a Mercer kernel function with a kernel bandwidth of $\sigma$. Let $\mathbb{E}[\cdot]$ be the mathematical expectation. Then, the correntropy can be defined as:

$$V(X,Y) = \mathbb{E}\left[k_\sigma(X,Y)\right]. \tag{1}$$

Generally, the Gaussian kernel is the most widely-used kernel in correntropy, and it is as follows:

$$k_\sigma(X,Y) = G_\sigma(e) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{e^2}{2\sigma^2}\right), \tag{2}$$

where $e = X - Y$ is the error value.

Here, a nonlinear mapping $\varphi(\cdot)$ is used by the kernel function to map the input space $\mathcal{U}$ to high-dimensional space $\mathcal{F}$, and it satisfies $\langle \varphi(x), \varphi(y) \rangle = k_\sigma(X, Y)$. Then, (1) is rewritten as:

$$V(X, Y) = E[\langle \varphi(X), \varphi(Y) \rangle]. \tag{3}$$

Since the joint probability density of data in practical applications is usually unknown, for a finite sample $\{x_i, y_i\}_{i=1}^{N}$, the correntropy can be defined as:

$$\widehat{V}(X, Y) = \frac{1}{N} \sum_{i=1}^{N} k_\sigma(x_i - y_i). \tag{4}$$

Generally, the kernel bandwidth is one of the key parameters in correntropy. Usually, a small kernel bandwidth makes the algorithm more robust to outliers, but it will lead to slow convergence and poor accuracy. On the other hand, when the kernel bandwidth increases, the robustness will be significantly reduced in the case of abnormal values. In order to achieve better performance, a new similarity measure MCC was proposed [11]. It can achieve faster convergence speed and higher filtering accuracy, while maintaining robustness to outliers. The mixture correntropy uses the mixture of two Gaussian functions as the kernel function, and its definition is as follows:

$$M(X, Y) = \mathbb{E}[\alpha G_{\sigma_1}(e) + (1 - \alpha) G_{\sigma_2}(e)], \tag{5}$$

where $0 \leqslant \alpha \leqslant 1$ is the mixture coefficient and $\sigma_1$ and $\sigma_2$ are the kernel bandwidths of the Gaussian functions $G_{\sigma_1}(\cdot)$ and $G_{\sigma_2}(\cdot)$, respectively. When the mixture coefficient $\alpha$ takes a suitable value, the performance of MCC can be better than that of the original correntropy criterion, so the mixture correntropy is a more flexible measure of similarity.

Typically, the empirical mixture correntropy loss can be expressed as $\widehat{L}(X, Y)$ or $\widehat{L}(e)$, where $X = [x_1, x_2, \cdots, x_N]^{\mathrm{T}}$, $Y = [y_1, y_2, \cdots, y_N]^{\mathrm{T}}$, and $e = [e_1, e_2, \cdots, e_N]^{\mathrm{T}}$. Then, it is defined as follows:

$$\begin{aligned} \widehat{L}(X, Y) &= 1 - \widehat{M}(X, Y) \\ &= 1 - \frac{1}{N} \sum_{i=1}^{N} \left[\alpha G_{\sigma_1}(e_i) + (1 - \alpha) G_{\sigma_2}(e_i)\right], \end{aligned} \tag{6}$$

where $e_i = x_i - y_i$.

Here, the Hessian matrix of $\widehat{L}(e)$ with respect to $e$ is:

$$\mathbf{H}_{\widehat{L}(e)}(e) = \left[ \frac{\partial^2 \widehat{L}(e)}{\partial e_i \partial e_j} \right] = \mathrm{diag}[\zeta_1, \zeta_2, \cdots, \zeta_N], \tag{7}$$

where $\zeta_i = \alpha \frac{\sigma_1^2 - e_i^2}{N \sigma_1^4} G_{\sigma_1}(e_i) + (1 - \alpha) \frac{\sigma_2^2 - e_i^2}{N \sigma_2^4} G_{\sigma_2}(e_i)$. Here, if $|e_i| \leqslant \sigma_1 \leqslant \sigma_2$, then $\zeta_i > 0$. For any point that satisfies $\|e\|_\infty \leqslant \sigma_1 \leqslant \sigma_2$, there is $\mathbf{H}_{\widehat{L}(e)}(e) \geqslant 0$.

It can be seen that the Hessian matrix $\mathbf{H}_{\widehat{L}(e)}$ of empirical mixture correntropy loss, as a function of $e$, is convex when the condition $\|e\|_\infty = \max_{1 \leqslant i \leqslant N} |e_i| \leqslant \sigma_1$ is satisfied. Therefore, the use of mixture correntropy as a loss function cannot be directly applied to convex optimization [33].

### *2.2. Kernel Conjugate Gradient Algorithm*

The CG is a specific method between the steepest descent and the Newton methods. It accelerates the typically slow convergence associated with the steepest descent, while avoiding the information requirements associated with the evaluation, storage, and inversion of Hessian as required by Newton's method [10].

Let $\mathbf{A} \in R^{n\times n}$ be a symmetric positive definite matrix and $\boldsymbol{d}_1, \boldsymbol{d}_2, \cdots, \boldsymbol{d}_m$ be a set of non-zero vectors in $R^n$. If $\boldsymbol{d}_i^{\mathrm{T}}\mathbf{A}\boldsymbol{d}_j = \mathbf{0}$ $(i \neq j)$, then we think that $\boldsymbol{d}_1, \boldsymbol{d}_2, \cdots, \boldsymbol{d}_m$ are conjugated to each other about $\mathbf{A}$.

There is an unconstrained optimization problem as follows:

$$\min_{x\in R^n} f(x), \tag{8}$$

where $f$ is a continuous differentiable function on $R^n$. The nonlinear method CG for solving the above (8) has the following iterative format:

$$\begin{aligned} x_{k+1} &= x_k + \alpha_k p_k, \\ p_k &= \begin{cases} -g_k, & k = 1 \\ -g_k + \beta_k p_{k-1}, & k \geqslant 2 \end{cases} \end{aligned} \tag{9}$$

where $g_k = \nabla f(x_k)$, $p_k$ is the search direction, $\alpha_k > 0$ is the step factor, $\beta_k$ is a certain parameter, and different $\beta_k$ corresponds to a different CG method. That is, when $f(x)$ is a strictly convex quadratic function, the search direction $p$ generated by the method (9) is conjugated to the Hessian matrix of $f(x)$. The process of CG iteration is described in Algorithm 1, where $\boldsymbol{r}_i$ is the residual vector, $\boldsymbol{p}_i$ is the search direction, $\boldsymbol{x}_i$ is the approximate solution. In addition, $\alpha_i$ and $\beta_i$ are the step factors, and the stopping criterion is that the algorithm achieves convergence.

**Algorithm 1** The conjugate gradient (CG) algorithm.

```
Input:
Given symmetric positive definite matrix $\mathbf{A}$;
Given the vector $\boldsymbol{b}$;
Given the initial iteration value $\boldsymbol{x}$;
Initialization:
$\boldsymbol{r}_0 = \boldsymbol{b} - \mathbf{A}\boldsymbol{x}_0$;  $\boldsymbol{p}_0 = \boldsymbol{r}_0$;
repeat
  for $k = 0, 1, \ldots$ do
    if $\boldsymbol{p}_k = 0$ then
      return $\boldsymbol{x}_0$;
    end else
      $\alpha_k = \frac{\boldsymbol{r}_k^{\mathrm{T}}\boldsymbol{r}_k}{\boldsymbol{p}_k^{\mathrm{T}}\mathbf{A}\boldsymbol{p}_k}$;
      $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k\boldsymbol{p}_k$;
      $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k\mathbf{A}\boldsymbol{p}_k$;
      $\beta_k = \frac{\boldsymbol{r}_{k+1}^{\mathrm{T}}\boldsymbol{r}_{k+1}}{\boldsymbol{r}_k^{\mathrm{T}}\boldsymbol{r}_k}$;
      $\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k\boldsymbol{p}_k$;
    end
  end
until Stopping_Criterion is met.
```

In order to address the nonlinear problem effectively, the KCG algorithm was proposed for online application [11]. In an effort to use kernel techniques, the solution vector of the algorithm CG is represented by a linear combination of input vectors. Then, the convergence speed of the online algorithm KCG is much faster than that of the algorithm KLMS. Actually, the KCG achieves the same convergence performance as the KRLS in many cases; however, the computational cost is greatly reduced [11]. Another attractive feature of KCG is that it does not require the user-defined parameters. The algorithm KCG is described as Algorithm 2. In this algorithm, $\mathbf{G}$ is the Gram matrix, $\boldsymbol{\eta}$ is the coefficient vector, $\kappa(\cdot,\cdot)$ is the Mercer kernel, $M$ is the size of the dictionary, $\boldsymbol{r}_0$ and $\boldsymbol{r}_1$ are the residual vectors, and $\boldsymbol{e}$ is the error vector. Moreover, $\alpha_1$, $\alpha_2$, and $\beta_2$ are step sizes; $\boldsymbol{S}_1$ is the residual vector of the normal equation; $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are intermediate vectors; and H stands for the conjugate transpose.

**Algorithm 2** The kernel conjugate gradient (KCG) algorithm.

```
Initialization:
$\mathbf{X}_1 = \boldsymbol{x}_1;\ \ q_1 = \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1);\ \ \boldsymbol{q}_1 = [\sqrt{q_1}];$
$\mathbf{G}_1 = q_1;\ \ \boldsymbol{\eta}_1 = \frac{\overline{d}_1}{q_1};\ \ \boldsymbol{e}_1 = 0;\ \ M = 1;$
repeat
  for $k = 2, 3, \ldots$ do
    $q_k = \kappa(\boldsymbol{x}_k, \boldsymbol{x}_k);$
    $g_k = [\kappa(\mathbf{X}_M(:,1), \boldsymbol{x}_k), \ldots, \kappa(\mathbf{X}_M(:,M), \boldsymbol{x}_k)];$
    $\nu_m = \frac{g_k(m)}{\sqrt{q_k \boldsymbol{q}_{k-1}(m)}};\ (m = 1, 2, \ldots, M)$
    if $\max\{|\nu_m|\} < \nu_0$ then
      $M = M + 1;$
      $\mathbf{X}_M = [\mathbf{X}_{M-1}, \boldsymbol{x}_k];\ \ \boldsymbol{q}_M = [\boldsymbol{q}_{M-1}, \sqrt{q_k}];$
      $\mathbf{G}_M = \begin{bmatrix} \mathbf{G}_{M-1} & g_k^{\mathrm{T}} \\ \overline{g}_k & q_k \end{bmatrix};$
      $\boldsymbol{r}_0 = [\boldsymbol{e}_{M-1}, \boldsymbol{d}_k - g_k \overline{\boldsymbol{\eta}}_{M-1}]^{\mathrm{H}};\ \ \boldsymbol{v}_1 = \mathbf{G}_M \boldsymbol{r}_0;$
      $\gamma_0 = \langle \boldsymbol{v}_1, \boldsymbol{r}_0 \rangle;\ \ \alpha_1 = \frac{\gamma_0}{\langle \boldsymbol{v}_1, \boldsymbol{v}_1 \rangle};$
      $\boldsymbol{r}_1 = \boldsymbol{r}_0 - \alpha_1 \boldsymbol{v}_1;\ \ \boldsymbol{s}_1 = \mathbf{G}_M \boldsymbol{r}_1;$
      $\gamma_1 = \langle \boldsymbol{s}_1, \boldsymbol{r}_1 \rangle;\ \ \beta_2 = \frac{\gamma_1}{\gamma_0};$
      $\boldsymbol{v}_2 = \beta_2 \boldsymbol{v}_1 + \boldsymbol{s}_1;\ \ \alpha_2 = \frac{\gamma_1}{\langle \boldsymbol{v}_2, \boldsymbol{v}_2 \rangle};$
      $\boldsymbol{\eta}_M = [\boldsymbol{\eta}_{M-1}, 0] + (\alpha_1 + \alpha_2 \beta_2)\, \boldsymbol{r}_0 + \alpha_2 \boldsymbol{r}_1;$
      $\boldsymbol{e}_M = (\boldsymbol{r}_1 - \alpha_2 \boldsymbol{v}_2)^{\mathrm{H}};$
    end
  end
until Stopping_Criterion is met.
```

Since the algorithm KCG is derived from the solution of the least squares problem, good performance can be maintained in a Gaussian noise environment [11]. However, in the non-Gaussian case, the performance of KCG may not be satisfactory [11]. Therefore, we used the mixture correntropy as the loss function and propose the algorithm KMCCG.

### *2.3. Sparsification Criterion*

KAF uses an online approaching strategy, that is each time a new set of data arrives, it is allocated a storage unit. The linear growth of the network structure leads to an increase in the memory requirements and calculations of KAF. Since the angle between two elements in the feature space can be represented by the inner product and the similarity of the elements in the space can be measured by an angle, then a simple sparsification criterion on the basis of the angle between elements in RKHS is used to control the network structure [11]. The cosine of the angle between $\varphi(\boldsymbol{x})$ and $\varphi(\boldsymbol{y})$ is as follows:

$$\nu(\boldsymbol{x}, \boldsymbol{y}) = \frac{\langle \varphi(\boldsymbol{x}), \varphi(\boldsymbol{y}) \rangle}{\|\varphi(\boldsymbol{x})\| \cdot \|\varphi(\boldsymbol{y})\|} = \frac{\kappa(\boldsymbol{y}, \boldsymbol{x})}{\sqrt{\kappa(\boldsymbol{x}, \boldsymbol{x})\kappa(\boldsymbol{y}, \boldsymbol{y})}}. \tag{10}$$

The algorithm reconstructs the network "dictionary" through the sparsification criterion. If the current dictionary is $D$ and a new sample $(\boldsymbol{x}_k, \boldsymbol{d}_k)$ is coming, the procedure of the angle criterion-based sparsification can be described through the following two steps.

First, the parameter $v$ is calculated:

$$\nu_k = \max_{1 \leqslant m \leqslant M} |\nu(\boldsymbol{x}_k, \tilde{\boldsymbol{x}}_m)| \in [0, 1]. \tag{11}$$

Second, if $\nu_k$ is smaller than the predefined threshold $\nu_0$, $(\varphi(\boldsymbol{x}_k), \boldsymbol{d}_k)$ is added to $D$, otherwise it is discarded. Because the parameter $\nu_0$ represents the level of similarity between the new element and those old elements in $D$, we call it the similarity parameter.

## 3. The Proposed Algorithm

### 3.1. Half-Quadratic Optimization of the Mixture Correntropy

For the mixture correntropy loss function (5), since its Hessian matrix is not positive definite, its global convexity cannot be guaranteed. Therefore, the mixture correntropy loss cannot be directly applied to the convex optimization problem. Fortunately, the half-quadratic (HQ) optimization method is an effective method to address the non-convex optimization problem [33], which converts the original objective function into the half-quadratic objective function. In this article, the HQ optimization method is used to transform the mixture correntropy loss function, and then, the method CG is employed to solve the transformation function.

Since the mixture correntropy is the sum of two exponential functions, we let $f(x) = \exp(-x)$ be an exponential function. The conjugate function of $f(x)$ is $g(v) = -v\ln(-v) + v$ $(v < 0)$. According to the theory of the conjugate function, the conjugate function of $g(v)$ is given by:

$$g^*(u) = \sup_{v<0}\{uv - g(v)\} = \sup_{v<0}\{uv + v\ln(-v) - v\}. \tag{12}$$

Let $f(v) = uv + v\ln(-v) - v$, where $v < 0$. Then, $f(v)$ reaches the maximum value $\exp(-u)$ at $v = -\exp(-u)$. Therefore, we can get:

$$g^*(u) = \sup_{v<0}\{uv + v\ln(-v) - v\} = \exp(-u), \tag{13}$$

where $v = -\exp(-u)$.

When $u = \frac{e_k^2}{2\sigma^2}$, we can get:

$$g^*\left(\frac{e_k^2}{2\sigma^2}\right) = \sup_{v<0}\left\{v\frac{e_k^2}{2\sigma^2} + v\ln(-v) - v\right\} = \exp\left(-\frac{e_k^2}{2\sigma^2}\right), \tag{14}$$

where $v = -\exp\left(-\frac{e_k^2}{2\sigma^2}\right)$.

Then, (5) can be written as:

$$\begin{aligned} M(X,Y) &= \frac{\alpha}{N}\sum_{i=1}^{N}\kappa_{\sigma_1}(x_i, y_i) + \frac{1-\alpha}{N}\sum_{i=1}^{N}\kappa_{\sigma_2}(x_i, y_i) \\ &= \frac{\alpha}{N}\sum_{i=1}^{N}\exp\left(-\frac{e_i^2}{2\sigma_1^2}\right) + \frac{1-\alpha}{N}\sum_{i=1}^{N}\exp\left(-\frac{e_i^2}{2\sigma_2^2}\right) \\ &= \frac{\alpha}{N}\sup_{v<0}\left\{v_i\frac{e_k^2}{2\sigma_1^2} - g(v_i)\right\} + \frac{1-\alpha}{N}\sup_{v'<0}\left\{v_i'\frac{e_k^2}{2\sigma_2^2} - g(v_i')\right\}. \end{aligned} \tag{15}$$

Because the solution to the mixture correntropy Loss (6) is equivalent to solving the following problem:

$$\max_{v<0} M(X,Y) = \frac{\alpha}{N}\max_{v<0}\left\{v_i\frac{e_k^2}{2\sigma_1^2} - g(v_i)\right\} + \frac{1-\alpha}{N}\max_{v'<0}\left\{v'_i\frac{e_k^2}{2\sigma_2^2} - g(v'_i)\right\}, \tag{16}$$

therefore, by solving the sum of the following weighted least squares problems, the equivalent solution of the mixture correntropy can be obtained.

$$\frac{\alpha}{N}\min\sum_{i=1}^{N}\left(-v_i\frac{e_i^2}{2\sigma_1^2}\right) + \frac{1-\alpha}{N}\min\sum_{i=1}^{N}\left(v'_i - \frac{e_i^2}{2\sigma_2^2}\right), \tag{17}$$

where $v_i = -\exp\left(-\frac{e_i^2}{2\sigma_1^2}\right)$, $v'_i = -\exp\left(-\frac{e_i^2}{2\sigma_2^2}\right)$, $e_i = y_i - f(x_i)$.

The Hessian matrix of the weighted least squares problem (17) is as follows:

$$\mathbf{H}(\boldsymbol{e}) = \operatorname{diag}\left[-\frac{\alpha v_1}{\sigma_1^2}, -\frac{\alpha v_2}{\sigma_1^2}, \ldots, -\frac{\alpha v_N}{\sigma_1^2}\right] + \operatorname{diag}\left[-\frac{(1-\alpha)v'_1}{\sigma_2^2}, -\frac{(1-\alpha)v'_2}{\sigma_2^2}, \ldots, -\frac{(1-\alpha)v'_N}{\sigma_2^2}\right], \tag{18}$$

where $v_i < 0$ and $v'_i < 0$. Hence, we obtain $\mathbf{H}(\boldsymbol{e}) > 0$, and then, (17) is a global convex optimization problem. Here, when the Hessian matrix is positive definite, it means that the function is a convex function. Then, the objective function can be optimized by the conjugated gradient method.

### *3.2. Kernel Mixture Correntropy Conjugate Gradient Algorithm*

The core of KAF is to transform the input data into a high-dimensional feature space through the kernel function. The inner product operation in the feature space is more efficiently calculated by the kernel technique. Its goal is to get the mapping function $f(x)$ between the input and output. According to the adaptive filtering theory, we can obtain:

$$f(\boldsymbol{x}_k) = \sum_{i=1}^{k}\eta_i\kappa\left(\cdot, \boldsymbol{x}_k\right), \tag{19}$$

where $\eta_i$ is the expansion coefficient and $\kappa\left(\cdot, \boldsymbol{x}_k\right)$ is a kernel function with the center $\boldsymbol{x}_k$.

According to (17), we consider the following kernel-induced weighted least squares problem:

$$\begin{aligned} &\frac{\alpha}{N}\min\sum_{i=1}^{k}\left(-\frac{v_i}{2\sigma_1^2}\left(d_i - \eta_i\kappa\left(\cdot, \boldsymbol{x}_k\right)\right)^2\right) + \frac{1-\alpha}{N}\min\sum_{i=1}^{k}\left(-\frac{v'_i}{2\sigma_2^2}\left(d_i - \eta_i\kappa\left(\cdot, \boldsymbol{x}_k\right)\right)^2\right) \\ =&\frac{\alpha}{N}\min\left\|\sqrt{\mathbf{V}}\left(\boldsymbol{d}^{\mathrm{T}} - \mathbf{G}_1\boldsymbol{\eta}\right)\right\|^2 + \frac{1-\alpha}{N}\min\left\|\sqrt{\mathbf{V}'}\left(\boldsymbol{d}^{\mathrm{T}} - \mathbf{G}_2\boldsymbol{\eta}\right)\right\|^2, \end{aligned} \tag{20}$$

where $\boldsymbol{\eta} = [\eta_1, \eta_2, \ldots, \eta_k]^{\mathrm{T}}$ is the expansion coefficient and $\mathbf{G}$ is the Gram matrix, which is defined as:

$$\mathbf{G}_k = \begin{bmatrix} \kappa\left(\boldsymbol{x}_1, \boldsymbol{x}_1\right) & \kappa\left(\boldsymbol{x}_1, \boldsymbol{x}_2\right) & \ldots & \kappa\left(\boldsymbol{x}_1, \boldsymbol{x}_k\right) \\ \kappa\left(\boldsymbol{x}_2, \boldsymbol{x}_1\right) & \kappa\left(\boldsymbol{x}_2, \boldsymbol{x}_2\right) & \ldots & \kappa\left(\boldsymbol{x}_2, \boldsymbol{x}_k\right) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa\left(\boldsymbol{x}_k, \boldsymbol{x}_1\right) & \kappa\left(\boldsymbol{x}_k, \boldsymbol{x}_2\right) & \ldots & \kappa\left(\boldsymbol{x}_k, \boldsymbol{x}_k\right) \end{bmatrix}. \tag{21}$$

Then, we use the method CG to solve the weighted least squares problem (20). The major work of the online algorithm KMCCG lies in the update of the Gram matrix $\mathbf{G}_M$ and the coefficient vector $\boldsymbol{\eta}_M$. The Gram matrix $\mathbf{G}_M$ can be updated as follows:

$$\mathbf{G}_M = \begin{bmatrix} \mathbf{G}_{M-1} & g_M^{\mathrm{T}} \\ \overline{g}_M & q_M \end{bmatrix}, \tag{22}$$

where $q_M = \kappa(\boldsymbol{x}_M, \boldsymbol{x}_M)$ and $g_M = [\kappa(\boldsymbol{x}_1, \boldsymbol{x}_M), \kappa(\boldsymbol{x}_2, \boldsymbol{x}_M), \ldots, \kappa(\boldsymbol{x}_{M-1}, \boldsymbol{x}_M)]$. Because $[\boldsymbol{\eta}_{M-1}, 0]^{\mathrm{T}}$ is a good approximation of $\boldsymbol{\eta}_M$, only a few iterations (one or two) with this initial vector can achieve satisfactory performance. We use $[\boldsymbol{\eta}_{M-1}, 0]^{\mathrm{T}}$ as the initial value of $\boldsymbol{\eta}_M$, and the initial residual $\boldsymbol{r}_0$ can be expressed as follows:

$$\begin{aligned} \boldsymbol{r}_0 &= \alpha \mathbf{V}_k \left( \boldsymbol{d}^{\mathrm{T}} - \mathbf{G}_k [\boldsymbol{\eta}_{M-1}, 0]^{\mathrm{T}} \right) + (1-\alpha)\mathbf{V}_k' \left( \boldsymbol{d}^{\mathrm{T}} - \mathbf{G}_k [\boldsymbol{\eta}_{M-1}, 0]^{\mathrm{T}} \right) \\ &= \alpha \mathbf{V}_k \left( \begin{bmatrix} \boldsymbol{d}_{k-1}^{\mathrm{T}} \\ d_k \end{bmatrix} - \begin{bmatrix} \mathbf{G}_{k-1} & g_k^{\mathrm{T}} \\ \overline{g}_k & q_k \end{bmatrix} \begin{bmatrix} \boldsymbol{\eta}_{k-1} \\ 0 \end{bmatrix} \right) + (1-\alpha)\mathbf{V}_k' \left( \begin{bmatrix} \boldsymbol{d}_{k-1}^{\mathrm{T}} \\ d_k \end{bmatrix} - \begin{bmatrix} \mathbf{G}_{k-1} & g_k^{\mathrm{T}} \\ \overline{g}_k & q_k \end{bmatrix} \begin{bmatrix} \boldsymbol{\eta}_{k-1} \\ 0 \end{bmatrix} \right) \\ &= \alpha \left[ \boldsymbol{e}_{k-1}, v_k (d_k - \overline{g}_k \boldsymbol{\eta}_{k-1}) \right]^{\mathrm{T}} + (1-\alpha) \left[ \boldsymbol{e}_{k-1}, v_k' (d_k - \overline{g}_k \boldsymbol{\eta}_{k-1}) \right]^{\mathrm{T}} \\ &= \alpha \left[ \boldsymbol{e}_{k-1}, v_k e_k \right]^{\mathrm{T}} + (1-\alpha) \left[ \boldsymbol{e}_{k-1}, v_k' e_k \right]^{\mathrm{T}}, \end{aligned} \tag{23}$$

where $\mathbf{V}_k = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0}^{\mathrm{T}} & v_k \end{bmatrix}$, $\mathbf{V}_k' = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0}^{\mathrm{T}} & v_k' \end{bmatrix}$, $v_k = \frac{1}{\sigma^2} \exp\left( -\frac{e_k^2}{2\sigma_1^2} \right)$, and $v_k' = \frac{1}{\sigma^2} \exp\left( -\frac{e_k^2}{2\sigma_2^2} \right)$. Then, we can get the KMCCG, as shown in Algorithm 3.

What follows is a working example we provide for this section to show the evolution of values throughout the whole process. Let $\boldsymbol{x}_1 = [0, 0, 0, 0, 1.0399]$, $\boldsymbol{d}_1 = [1.5700]$, $\boldsymbol{x}_2 = [0, 0, 0, 1.0399, 1.5700]$, and $\boldsymbol{d}_2 = [1.3156]$. Then, according to Algorithm 3, the evolution of the values is as follows.

When $k = 2$, $q_2 = \kappa(x_2, x_2) = 0.6595$, $g_2 = \kappa(x_1, x_2) = 0.3337$, and $v_1 = 0.3337$, then because $\max\{|v_1|\} < v_0 = 0.8$, $\boldsymbol{x}_2$ is added to the dictionary, which means that $M = M + 1 = 2$ and $\mathbf{X}_2 = [\mathbf{X}_1, \boldsymbol{x}_2]$. The Gram matrix $\mathbf{G}_M$ can be updated as follows:

$$\mathbf{G}_2 = \begin{bmatrix} \mathbf{G}_1 & g_2^{\mathrm{T}} \\ \overline{g}_2 & q_2 \end{bmatrix}. \tag{24}$$

Then, the residual can be obtained as $\boldsymbol{r}_0 = [0, 0.3991]^{\mathrm{T}}$ and $\boldsymbol{r}_1 = [-0.1834, 0.1210]^{\mathrm{T}}$. On the basis of this, the correlation coefficient $\boldsymbol{\eta}_2$ can be updated. When the new input arrives, the algorithm will continuously update the correlation coefficient to make it more suitable for the next input. Finally, the algorithm will stop when the convergence condition is satisfied, that is it reaches the maximum number of iterations.

**Algorithm 3** The kernel mixture correntropy conjugate gradient (KMCCG) algorithm.

```
Initialization:
$\mathbf{X}_1 = \boldsymbol{x}_1;\ \ q_1 = \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1);\ \ \boldsymbol{q}_1 = [\sqrt{q_1}];$
$\mathbf{G}_1 = q_1;\ \ \boldsymbol{\eta}_1 = \frac{\bar{d}_1}{q_1};\ \ \boldsymbol{e}_1 = 0;\ \ M = 1;$
repeat
  for $k = 2, 3, \ldots$ do
    $q_k = \kappa(\boldsymbol{x}_k, \boldsymbol{x}_k);$
    $g_k = [\kappa(\mathbf{X}_M(:,1), \boldsymbol{x}_k), \ldots, \kappa(\mathbf{X}_M(:,M), \boldsymbol{x}_k)];$
    $\nu_m = \frac{g_k(m)}{[\sqrt{q_k}\boldsymbol{q}_{k-1}(m)]};\ \ (m = 1, 2, \ldots, M)$
    if $\max\{|\nu_m|\} < \nu_0$ then
      $M = M + 1;$
      $\mathbf{X}_M = [\mathbf{X}_{M-1}, \boldsymbol{x}_k];\ \ \boldsymbol{q}_M = [\boldsymbol{q}_{M-1}, \sqrt{q_k}];$
      $\mathbf{G}_M = \begin{bmatrix} \mathbf{G}_{M-1} & g_k^{\mathrm{T}} \\ \bar{g}_k & q_k \end{bmatrix};$
      $e_k = d_k - \bar{g}_k \boldsymbol{\eta}_{k-1};$
      $v_k = \frac{1}{\sigma_1^2} \cdot \exp\left(-\frac{e_k^2}{2\sigma_1^2}\right);\ \ v_k' = \frac{1}{\sigma_2^2} \cdot \exp\left(-\frac{e_k^2}{2\sigma_2^2}\right);$
      $\boldsymbol{r}_0 = \alpha\left[\boldsymbol{e}_{k-1}, v_k e_k\right]^{\mathrm{T}} + (1-\alpha)\left[\boldsymbol{e}_{k-1}, v_k' e_k\right]^{\mathrm{T}};\ \ \boldsymbol{v}_1 = \mathbf{G}_M \boldsymbol{r}_0;$
      $\gamma_0 = \langle \boldsymbol{v}_1, \boldsymbol{r}_0 \rangle;\ \ \alpha_1 = \frac{\gamma_0}{\langle \boldsymbol{v}_1, \boldsymbol{v}_1 \rangle};$
      $\boldsymbol{r}_1 = \boldsymbol{r}_0 - \alpha_1 \boldsymbol{v}_1;\ \ \boldsymbol{s}_1 = \mathbf{G}_M \boldsymbol{r}_1;$
      $\gamma_1 = \langle \boldsymbol{s}_1, \boldsymbol{r}_1 \rangle;\ \ \beta_2 = \frac{\gamma_1}{\gamma_0};$
      $\boldsymbol{v}_2 = \beta_2 \boldsymbol{v}_1 + \boldsymbol{s}_1;\ \ \alpha_2 = \frac{\gamma_1}{\langle \boldsymbol{v}_2, \boldsymbol{v}_2 \rangle};$
      $\boldsymbol{\eta}_M = [\boldsymbol{\eta}_{M-1}, 0] + (\alpha_1 + \alpha_2\beta_2)\boldsymbol{r}_0 + \alpha_2 \boldsymbol{r}_1;$
      $\boldsymbol{e}_M = (\boldsymbol{r}_1 - \alpha_2 \boldsymbol{v}_2)^{\mathrm{H}};$
    end
  end
until Stopping_Criterion is met.
```

## 3.3. Computational Time Complexity Analysis

As shown in Table 1, we obtain the computational time complexity of four algorithms KLMS, KRLS, KCG, and KMCCG, after analyzing the implementation process of those algorithms. In this table, M is the dictionary size in the algorithm. KLMS is a simple KAF with minimal computational complexity. KRLS requires $(4\mathrm{M}^2 + 4\mathrm{M})$ additions, $(4\mathrm{M}^2 + 4\mathrm{M})$ multiplications, and one division per iteration, while KCG achieves a convergence speed and filtering accuracy comparable to KRLS [11], and its computational complexity is relatively small. In addition, compared with KCG, KMCCG requires two divisions and one multiplication when calculating $v_k$ and also requires an addition and a multiplication to update the residual vector $\boldsymbol{r}_0$. Table 1 shows that in each iteration, KMCCG requires fewer additions and multiplications than KRLS, but requires four more division operations. Because the number of instruction cycles required by the division operation is generally 20-times that of the addition operation and M is usually greater than 100, the computational complexity of KMCCG is still lower than that of the KRLS algorithm. Moreover, when the input vector does not meet the sparsification criterion, there is no additional calculation for KMCCG, but there are still $(4\mathrm{M}^2 + 4\mathrm{M})$ additions and $(4\mathrm{M}^2 + 4\mathrm{M})$ multiplications for KRLS. Therefore, KMCCG can achieve higher prediction accuracy with less computational and storage costs.

**Table 1.** Computational cost for dictionary update. KLMS (kernel least mean squares); KRLS (kernel recursive least squares); KCG (kernel conjugate gradient); KMCCG (kernel mixture correntropy conjugate gradient).

| **Algorithm** | **Additions** | **Multiplications** | **Divisions** |
|---|---|---|---|
| KLMS | M | M | 0 |
| KRLS | $4M^2 + 4M$ | $4M^2 + 4M$ | 1 |
| KCG | $2M^2 + 8M$ | $2M^2 + 10M$ | 3 |
| KMCCG | $2M^2 + 8M + 1$ | $2M^2 + 10M + 2$ | 5 |

## 4. Experimental Results and Discussions

In this section, the experiments on short-term predictions of the Mackey–Glass chaotic time series, minimum daily temperatures time series, and the real-world malware API call sequence are conducted to illustrate the performance of our proposed algorithm.

### *4.1. Mackey–Glass Time Series Prediction*

The chaotic time series is one of the fundamental forms of movement in nature and human society. Generally, the classical Mackey–Glass chaotic time series is generated by the following differential equation [6]:

$$\frac{dx(t)}{dt} = -bx(t) + \frac{ax(t-\tau)}{1 + x(t-\tau)^n}, \tag{25}$$

where the parameters are set to $a = 0.2, b = 0.1, n = 10$, and $\tau = 30$. Moreover, the sampling period is 6 s. This experiment uses the past seven samples $\boldsymbol{u}(k) = [x(k), x(k-1), \ldots, x(k-6)]$ to predict the current input $\boldsymbol{d}(k) = x(k+1)$. Then, we use the dimensions of the matrix to represent the size of the matrix. Therefore, the size of the training input set is $(7 \times 1000)$, the size of the training target set is $(1000 \times 1)$, the size of the testing input set is $(7 \times 200)$, and the size of the testing target set is $(200 \times 1)$.

The algorithm KMCCG was compared with the quantized KLMS (QKLMS) algorithm [34], the quantized kernel maximum correntropy (QKMC) algorithm [35], and the kernel maximum mixture correntropy (KMMCC) algorithm [15] in four different noise environments, to verify the performance of our proposed algorithm. Here, QKLMS is one of the most classical KAF algorithms based on the mean square error (MSE) criterion, and it achieves good prediction accuracy in the Gaussian noise environment. QKMC is a KAF algorithm based on correntropy, which can also achieve good prediction accuracy and maintain robustness. The recently-proposed algorithm KMMCC combined with the mixture correntropy criterion has been demonstrated to be able to obtain satisfactory prediction accuracy and robustness. All algorithms were configured with a Gaussian kernel. For a fair comparison, the optimal parameter setting was conducted to let each algorithm achieve the desirable performance. Finally, the performance of the algorithm was evaluated by MSE, which is defined here as follows:

$$\text{MSE} = 10 \log_{10} \left( \frac{1}{N} \sum_{i=1}^{N} (d(i) - y(i))^2 \right), \tag{26}$$

where $N$ represents the number of predicted values.

Figure 1 shows the learning performance of these algorithms in four noise environments. Obviously, in all four types of noise environments, the testing MSE of KMCCG was smaller than that of the stochastic gradient-based filtering algorithms, i.e., QKLMS, QKMC, and KMMCC. Meanwhile, the convergence speed of KMCCG was obviously faster than that of the other compared algorithms. This verifies that the CG technique used in KMCCG can achieve a faster convergence speed and higher learning accuracy. Therefore, the algorithm KMCCG can achieve the best performance in all the compared algorithms.

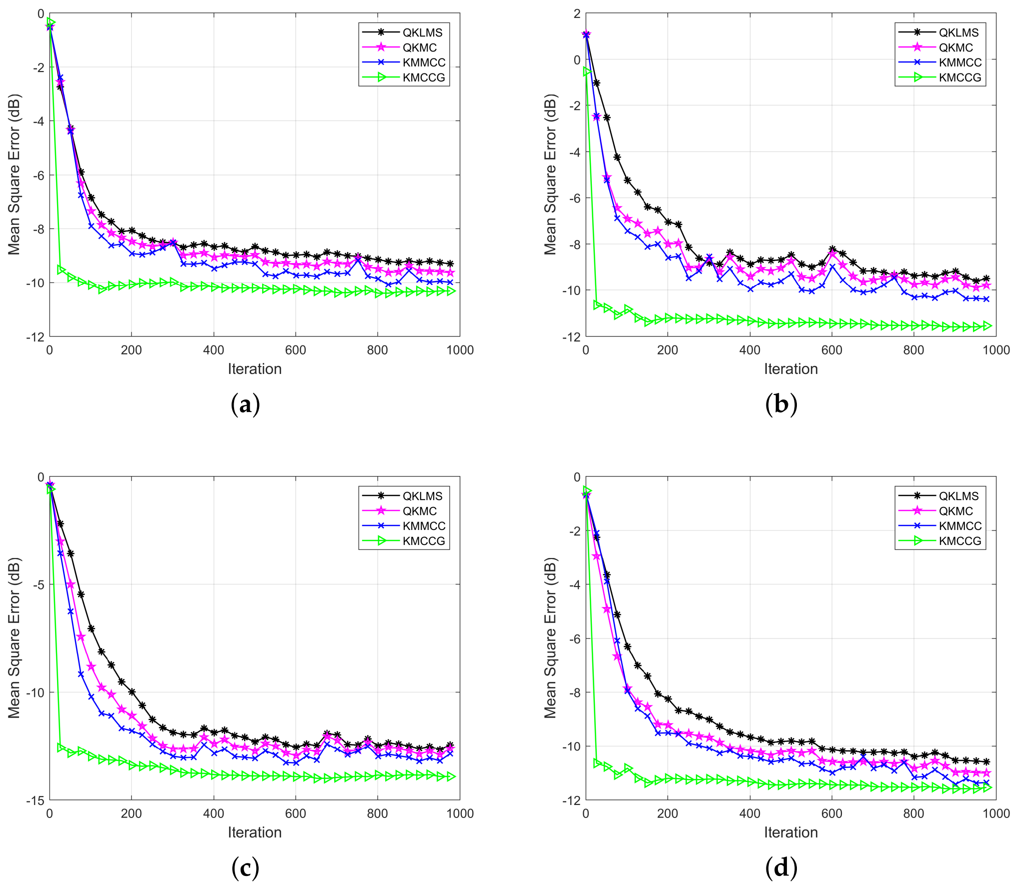


**Figure 1.** Mackey–Glass time series prediction: learning performance in terms of the testing MSE (mean square error) for QKLMS (quantized kernel least mean squares), QKMC (quantized kernel maximum correntropy), KMMCC (kernel maximum mixture correntropy), and KMCCG (kernel mixture correntropy conjugate gradient) under different noise environments: (**a**) Gaussian; (**b**) Bernoulli; (**c**) sine wave; (**d**) uniform.

## 4.2. Minimum Daily Temperatures Time Series Prediction

In this section, the minimum daily temperatures time series is selected as the dataset to verify the performance of the proposed algorithm. This dataset describes the minimum daily temperature in Melbourne, Australia, for 10 years (1981–1990) [36]. The unit is Celsius, and there are 3650 observations. The data source is the Australian Meteorological Agency.

Here, we use the previous five input samples $\boldsymbol{x}(k) = [x(k-5), x(k-4), \ldots, x(k-1)]^{\mathrm{T}}$ to predict the current point $\boldsymbol{d}(k) = x(k)$, where $\boldsymbol{x}(k)$ and $\boldsymbol{d}(k)$ represent the input vector and the corresponding expected output, respectively. Additionally, the size of the training input set as (5×1000); the size of the training target set was (1000×1); the size of the test input set was (5×200); and the size of the test target set was (200×1). Finally, the MSE was also used to evaluate the performance of those algorithms. Then, our algorithms was compared with QKLMS, QKMC, and KMMCC to verify the computational performance.

Figure 2 shows the learning curve for these algorithms. Obviously, the testing MSE of KMCCG is less than that of QKLMS, QKMC, and KMMCC, which demonstrates that the proposed algorithm can perform better than all three other algorithms.

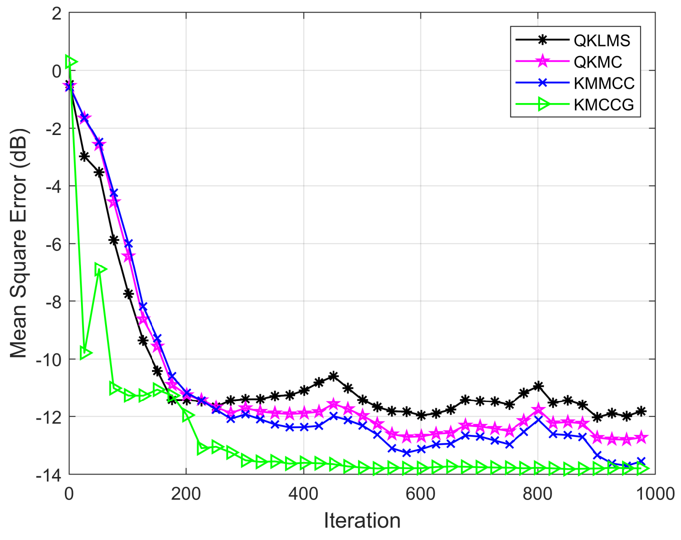


**Figure 2.** Minimum daily temperatures time series prediction: learning performance in terms of the testing MSE for QKLMS, QKMC, KMMCC, and KMCCG.

### *4.3. Malware API Call Sequence Prediction*

In this section, We apply the proposed algorithm to the malware API call sequence prediction, while verifying the effectiveness of our algorithm through the actual time series data. The purpose of this experiment is to predict what the next API would be, which can be used to determine whether it is malware or not.

#### 4.3.1. Background

API is the service interface provided by the operating system. Applications call the API when completing file reading and writing, network access, and other tasks [37]. Meanwhile, malware also needs to call the API when implementing functions. Hence, it is an effective method to predict and detect malware behavior by extracting the API call sequence [21].

With the rapid advances in computational intelligence methodology, using machine learning algorithms to predict malware via the API call sequence can make the malware prediction more intelligent, and the new malware can be detected in a more timely manner [38]. In this field, SVM, ANN, and other methods have been applied to malware prediction and detection, and some satisfactory results are achieved.

In [39], with the help of global features using the Gabor wavelet transform and Gist, the feed-forward ANN was developed to identify the behavior of malicious data with a good accuracy. In [40], after abstracting the complex behaviors based on the semantic analysis of dynamic API sequences, an SVM was proposed to achieve malware detection with good generalization ability. Furthermore, with the popular use of the deep learning method, some DNN models were also applied to tackle the issue of malware detection. For instance, in [41], the features were extracted from five minutes of API call log sequences by using a recurrent neural network, and then, they were input to the convolutional neural network to achieve deep learning with the purpose of malware detection.

Although some good performances have been achieved by using the above approaches, there still exist several limitations, such as the long training time and difficulty in parameter determination. Since mixture correntropy as a new measure of local similarity defined in kernel space can be used to address large outliers, hence, in order to reduce the training time while maintaining high prediction accuracy and robustness to abnormal data in the API call sequence, our algorithm KMCCG can be considered to cope with the malware prediction. Here, it should be noted that although some traditional machine learning-based malware prediction and detection algorithms may be vulnerable to adversarial methods or tools, such as EvadeML [42] and poisoning attack [43], the algorithm KMCCG may be a better choice in malware prediction, in consideration of the satisfactory robustness achieved by using mixture correntropy.

We mainly analyzed the acquired API call sequence and predicted the malicious behavior that may occur in the future using our proposed kernel learning algorithm. Then, through the combination of these predicted malicious behaviors with the actual detected malicious behaviors, we will extract feature vectors and integrate them as the discriminant basis of malware detection. In so doing, we can determine whether the application belongs to malware or not, through the machine learning classification model.

### 4.3.2. Experimental Result

API call information can be extracted by static and dynamic methods. Through the use of the static method [44], the API list can be extracted from the portable executable (PE) format of the executable files. Furthermore, with the dynamic method [45], the called API can be observed by running the executable files.

While creating the dataset, we randomly selected Windows malware samples from the malware datasets of Dasmalwerk and VirusShare and put the software into the cuckoo sandbox to analyze the report automatically. In order to avoid related security issues caused by malware propagation and accidental execution, here we chose to deploy the cuckoo sandbox to the Ubuntu environment. Figure 3 shows a flowchart for building the malware corpus. The specific analysis process of the sample is as follows: (1) The first step is to launch cuckoo on the Linux platform. (2) Then, we submit the sample to be analyzed to cuckoo. (3) Cuckoo uploads the sample to the virtual machine and collects the behavior data. (4) After the analysis is completed, cuckoo will generate an analysis report in its own working directory. Then, the API sequence is extracted from the report, and Figure 4 shows some API call time series. Each line in this figure represents a malware API call time series. In the following experiment, the API call sequence of a certain malware sample is selected as the dataset.

This experiment used the past seven samples to predict the current input. The size of the training input set was (7 × 1000); the size of the training target set was (1000 × 1), the size of the testing input set was (7 × 200), and the size of testing target set was (200 × 1). Figure 5 is the time series dataset obtained by replacing the API call sequence with the word frequency for which the API appeared in the whole dataset. Figure 6 shows the normalized sequence shown in Figure 5.

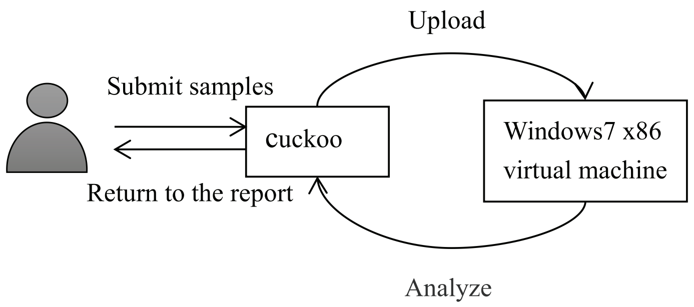
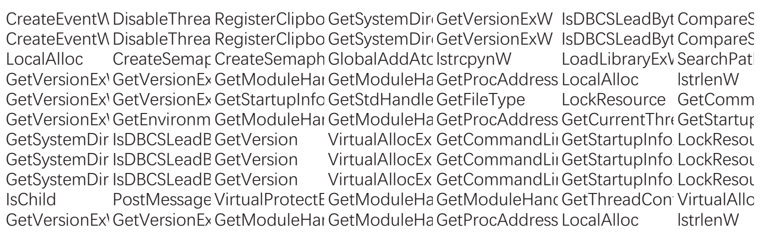


**Figure 3.** Flowchart for building a malware corpus.



| | | | | | | |
|---|---|---|---|---|---|---|
| CreateEventW | DisableThrea | RegisterClipbo | GetSystemDir | GetVersionExW | IsDBCSLeadByt | CompareS |
| CreateEventW | DisableThrea | RegisterClipbo | GetSystemDir | GetVersionExW | IsDBCSLeadByt | CompareS |
| LocalAlloc | CreateSemap | CreateSemaph | GlobalAddAtc | lstrcpynW | LoadLibraryExV | SearchPat |
| GetVersionEx\ | GetVersionEx | GetModuleHa | GetModuleHa | GetProcAddress | LocalAlloc | lstrlenW |
| GetVersionEx\ | GetVersionEx | GetStartupInfo | GetStdHandle | GetFileType | LockResource | GetComm |
| GetVersionEx\ | GetEnvironm | GetModuleHa | GetModuleHa | GetProcAddress | GetCurrentThre | GetStartup |
| GetSystemDir | IsDBCSLeadB | GetVersion | VirtualAllocEx | GetCommandLi | GetStartupInfo | LockResou |
| GetSystemDir | IsDBCSLeadB | GetVersion | VirtualAllocEx | GetCommandLi | GetStartupInfo | LockResou |
| GetSystemDir | IsDBCSLeadB | GetVersion | VirtualAllocEx | GetCommandLi | GetStartupInfo | LockResou |
| IsChild | PostMessage | VirtualProtectE | GetModuleHa | GetModuleHanc | GetThreadCon | VirtualAllc |
| GetVersionEx\ | GetVersionEx | GetModuleHa | GetModuleHa | GetProcAddress | LocalAlloc | lstrlenW |

**Figure 4.** Malware API (application programming interface) call time series report.

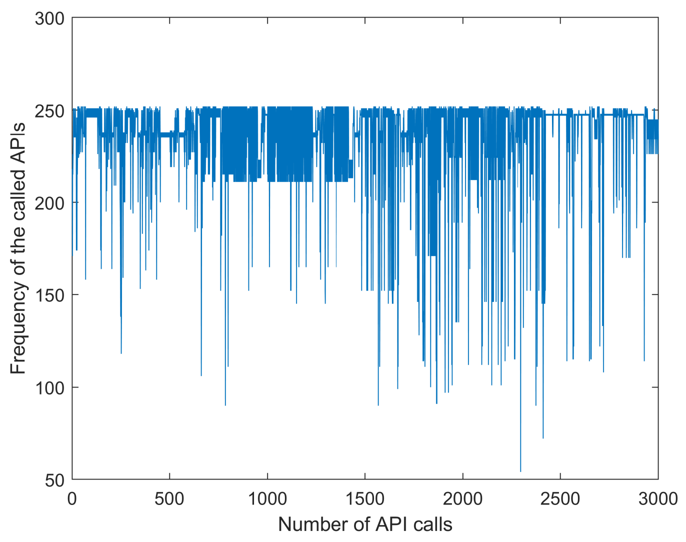


**Figure 5.** Original malware API call time series.

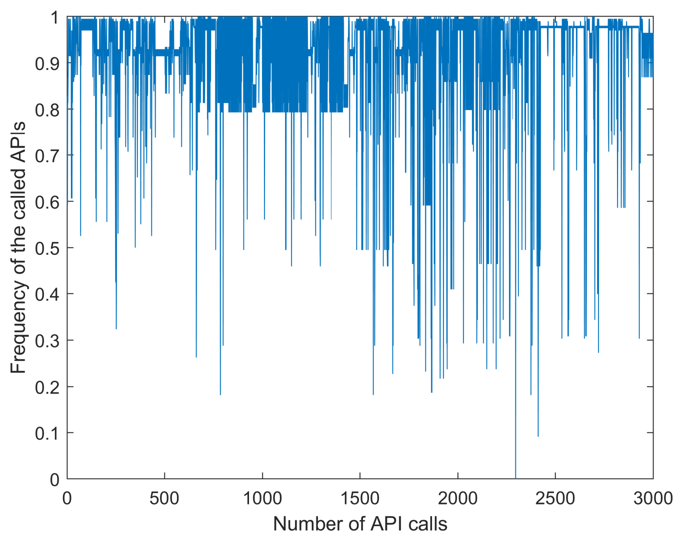


**Figure 6.** Normalized malware API call time series.

In this experiment, the performance of the algorithm is verified by comparing the prediction accuracy of QKLMS, KMMCC, ANN, SVM, and KMCCG. Considering that the prediction error is an effective evaluation metric, we still adopted the transformed error (26) to evaluate the algorithm performance. Figure 7 shows the learning performance of all five algorithms, which represents the relationship between the testing MSE of the algorithm and the number of iterations. Obviously, the MSE of KMCCG is smaller than that of the classical KAF algorithms, i.e., QKLMS and KMMCC. The total training time and the average value of MSE for five runs of the experiment are summarized in Table 2. It can be found that the proposed algorithm can achieve a prediction accuracy equivalent to the popular ANN and SVM, but spent less training time. Here, the algorithm KMCCG can be successfully applied to predict malware API call sequences, which verifies the satisfactory performance of our proposed algorithm.

Then, the time series of malware API calls were combined with Gaussian noise to further verify the robustness of the algorithm. In the real world, noise is often not caused by a single source, but a combination of many different sources. As the number of noise sources increases, it tends to a Gaussian distribution. Here, Gaussian noise is considered in the experiment to analyze the impact of noise in the system. Figure 8 shows the performance comparison of the proposed algorithm with the other four algorithms in the Gaussian noise environment. The evaluation metric is also the MSE shown in (26). It can be seen that the algorithm KMCCG can achieve higher prediction accuracy and be more stable in the noise environment. This shows that KMCCG has satisfactory robustness.

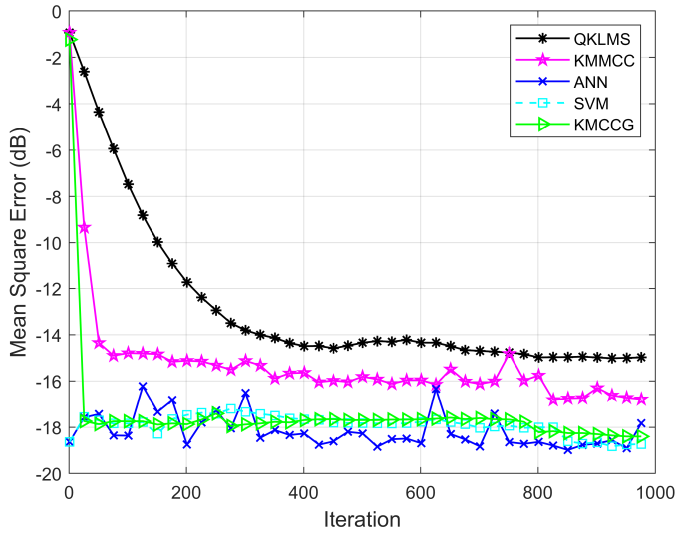


**Figure 7.** Testing MSEs of QKLMS, KMMCC, ANN (artificial neural network), SVM (support vector machine), and KMCCG for malware API call time series prediction.

**Table 2.** Computational results of QKLMS, KMMCC, ANN, SVM, and KMCCG in API call time series prediction.

| **Algorithm** | **Time (s)** | **MSE (dB)** |
|---|---|---|
| QKLMS | 39.03 | −12.9139 |
| KMMCC | 52.46 | −15.4081 |
| ANN | 841.41 | −18.1565 |
| SVM | 106.21 | −17.8825 |
| KMCCG | 2.24 | −17.8421 |

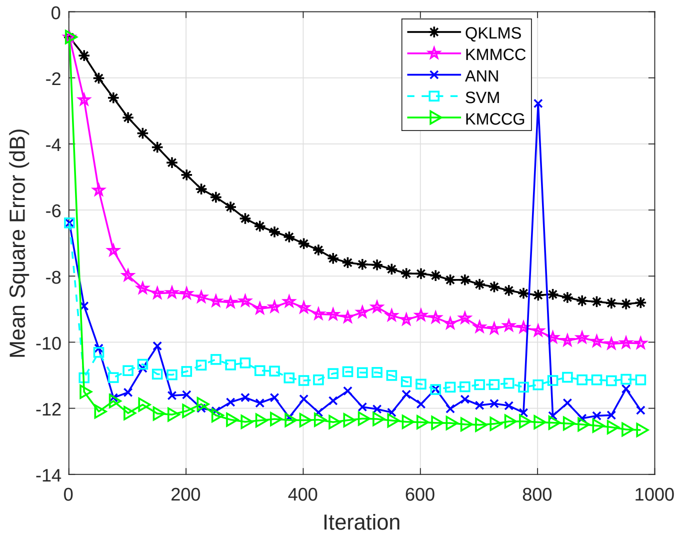


**Figure 8.** Testing MSEs of QKLMS, KMMCC, ANN, SVM, and KMCCG for malware API call time series prediction in the noise environment.

## 5. Conclusions

Through the combination of the MCC and the algorithm KCG, a novel kernel learning algorithm, i.e., KMCCG, is proposed in this article. Specifically, in an effort to curb effectively the growing RBF network in our algorithm KMCCG, a sparsification criterion based on the angle between elements in RKHS is used to control the increase of data size in online applications, which is equivalent to the coherence criterion. The proposed algorithm achieves much faster convergence speed than the algorithm KLMS and lower computational complexity than the algorithm KRLS. The prediction results for Mackey–Glass chaotic time series and Lorentz time series showed that our algorithm achieved good performance in robustness, filtering accuracy, and computational efficiency. Furthermore, the proposed kernel learning algorithm was applied to malware prediction. The results also showed that the algorithm KMCCG not only had a short training time, but also maintained a high prediction accuracy, which further verified the satisfactory performance of KMCCG.

As a use case of our algorithm, this article only focused on the task of malware API call sequence prediction. Actually, the prediction experiment of malware API call time series is only a part of the malware detection technology, and the software cannot be directly classified as malware or a benign one by only using our method. In future work, we will combine the results of future behavior prediction with the actual detected malicious behavior as the basis of classification reference and judge whether the application belongs to malware or not. Moreover, to further extend the applications of malware prediction and detection while using the algorithm KMCCG, we will discuss some other classification tasks for malware through the evaluation of false positives.

**Author Contributions:** In this article, X.L. and Y.G. provided the original ideas and were responsible for revising the whole article; N.X. designed and performed the experiments and wrote the original article; W.W., L.W., C.H., and W.Z. analyzed the data. All authors read and approved the final manuscript.

**Funding:** This research is funded by the National Natural Science Foundation of China under Grants U1836106 and U1736117, the National Key Research and Development Program of China under Grant 2018YFC0808306, and the USTB–NTUT Joint Research Program under Grant TW201705.

**Acknowledgments:** The authors would like to thank the Editorial board and reviewers for the improvement of this article.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Firdausi, I.; Lim, C.; Erwin, A.; Nugroho, A.S. Analysis of machine learning techniques used in behavior-based malware detection. In Proceedings of the Second International Conference on Advances in Computing, Control, and Telecommunication Technologies, Jakarta, Indonesia, 2–3 December 2010; pp. 201–203.
2. Xiao, X.; Zhang, S.; Mercaldo, F.; Hu, G.; Sangaiah, A.K. Android malware detection based on system call sequences and LSTM. *Multimed. Tools Appl.* **2019**, *78*, 3979–3999. [CrossRef]
3. Luo, X.; Jiang, C.; Wang, W.; Xu, Y.; Wang, J.H.; Zhao, W. User behavior prediction in social networks using weighted extreme learning machine with distribution optimization. *Future Gener. Comput. Syst.* **2019**, *93*, 1023–1035. [CrossRef]
4. Han, M.; Zhang, S.; Xu, M.; Qiu, T.; Wang, N. Multivariate chaotic time series online prediction based on improved kernel recursive least squares algorithm. *IEEE Trans. Cybern.* **2019**, *49*, 1160–1172. [CrossRef]
5. Sahoo, D.; Hoi, S.C.H.; Li, B. Large scale online multiple kernel regression with application to time-series prediction. *ACM Trans. Knowl. Discov. Data* **2019**, *13*, 9. [CrossRef]
6. Liu, W.; Príncipe, J.C.; Haykin, S. *Kernel Adaptive Filtering*; Wiley: Hoboken, NJ, USA, 2011.
7. Liu, W.; Pokharel, P.P.; Príncipe, J.C. The kernel least-mean-square algorithm. *IEEE Trans. Signal Process.* **2008**, *56*, 543–554. [CrossRef]
8. Engel, Y.; Mannor, S.; Meir, R. The kernel recursive least-squares algorithm. *IEEE Trans. Signal Process.* **2004**, *52*, 2275–2285. [CrossRef]
9. Luo, X.; Deng, J.; Liu, J.; Wang, W.; Ban, X.; Wang, J.H. A quantized kernel least mean square scheme with entropy-guided learning for intelligent data analysis. *China Commun.* **2017**, *14*, 127–136. [CrossRef]

10. Ahmad, N.A. A globally convergent stochastic pairwise conjugate gradient-based algorithm for adaptive filtering. *IEEE Signal Process. Lett.* **2008**, *15*, 914–917. [CrossRef]
11. Zhang, M.; Wang, X.; Chen, X.; Zhang, A. The kernel conjugate gradient algorithms. *IEEE Trans. Signal Process.* **2018**, *66*, 4377–4387. [CrossRef]
12. Gunduz, A.; Príncipe, J.C. Correntropy as a novel measure for nonlinearity tests. *Signal Process.* **2009**, *89*, 14–23. [CrossRef]
13. Chen, M.; Li, Y.; Luo, X.; Wang, W.; Wang, L.; Zhao, W. A novel human activity recognition scheme for smart health using multilayer extreme learning machine. *IEEE Internet Things J.* **2019**, *6*, 1410–1418. [CrossRef]
14. Sun, J.; Wang, Z.; Luo, X.; Shi, P.; Wang, W.; Wang, L.; Wang, J.H.; Zhao, W. A parallel recommender system using a collaborative filtering algorithm with correntropy for social networks. *IEEE Trans. Netw. Sci. Eng.* **2018**. [CrossRef]
15. Chen, B.; Wang, X.; Lu, N.; Wang, S.; Cao, J.; Qin, J. Mixture correntropy for robust learning. *Pattern Recogn.* **2018**, *79*, 318–327. [CrossRef]
16. Fan, H.; Song, Q. A linear recurrent kernel online learning algorithm with sparse updates. *Neural Netw.* **2014**, *50*, 142–153. [CrossRef]
17. Platt, J. *A Resource-allocating Network for Function Interpolation*; MIT Press: Cambridge, MA, USA, 1991.
18. Liu, W.; Park, I.; Príncipe, J.C. An information theoretic approach of designing sparse kernel adaptive filters. *IEEE Trans. Neural Netw.* **2009**, *20*, 1950–1961. [CrossRef]
19. Teng, H.; Liu, Y.; Liu, A.; Xiong, N.N.; Cai, Z.; Wang, T.; Liu, X. A novel code data dissemination scheme for Internet of Things through mobile vehicle of smart cities. *Future Gener. Comput. Syst.* **2019**, *94*, 351–367. [CrossRef]
20. Rhode, M.; Burnap, P.; Jones, K. Early-stage malware prediction using recurrent neural networks. *Comput. Secur.* **2018**, *77*, 578–594. [CrossRef]
21. Onwuzurike, L.; Mariconti, E.; Andriotis, P.; De Cristofaro, E.; Ross, G.; Stringhini, G. Mamadroid: Detecting android malware by building Markov chains of behavioral models. *ACM Trans. Priv. Secur.* **2019**, *22*, 14. [CrossRef]
22. Príncipe, J.C. *Information Theoretic Learning: Renyi's Entropy and Kernel Perspectives*; Springer: New York, NY, USA, 2010.
23. Zhang, J.F.; Qiu, T.S. A robust correntropy based subspace tracking algorithm in impulsive noise environments. *Digital Signal Process. Rev. J.* **2017**, *62*, 168–175. [CrossRef]
24. Luo, X.; Xu, Y.; Wang, W.; Yuan, M.; Ban, X.; Zhu, Y.; Zhao, W. Towards enhancing stacked extreme learning machine with sparse autoencoder by correntropy. *J. Franklin Inst.* **2018**, *355*, 1945–1966. [CrossRef]
25. Du, B.; Tang, X.; Wang, Z.; Zhang, L.; Tao, D. Robust graph-based semisupervised learning for noisy labeled data via maximum correntropy criterion. *IEEE Trans. Cybern.* **2019**, *49*, 1440–1453. [CrossRef]
26. Luo, X.; Sun, J.; Wang, L.; Wang, W.; Zhao, W.; Wu, J.; Wang, J.H.; Zhang, Z. Short-term wind speed forecasting via stacked extreme learning machine with generalized correntropy. *IEEE Trans. Ind. Inf.* **2018**, *14*, 4963-4971. [CrossRef]
27. Chen, B.; Xing, L.; Liang, J.; Zheng, N.; Príncipe, J.C. Steady-state mean-square error analysis for adaptive filtering under the maximum correntropy criterion. *IEEE Signal Process. Lett.* **2014**, *21*, 880–884.
28. Wu, Z.Z.; Peng, S.Y.; Chen, B.D.; Zhao, H.Q. Robust Hammerstein adaptive filtering under maximum correntropy criterion. *Entropy* **2015**, *17*, 7149–7166. [CrossRef]
29. Peng, S.; Chen, B.; Sun, L.; Ser, W.; Lin, Z. Constrained maximum correntropy adaptive filtering. *Signal Process.* **2017**, *140*, 116–126. [CrossRef]
30. Lu, L.; Zhao, H. Active impulsive noise control using maximum correntropy with adaptive kernel size. *Mech. Syst. Signal Process.* **2017**, *87*, 180–191. [CrossRef]
31. Zhang, Z.Y.; Qiu, J.Z.; Ma, W.T. Adaptive extended Kalman filter with correntropy loss for robust power system state estimation. *Entropy* **2019**, *21*, 293. [CrossRef]
32. He, Y.; Wang, F.; Wang, S.; Cao, J.; Chen, B. Maximum correntropy adaptation approach for robust compressive sensing reconstruction. *Inf. Sci.* **2019**, *480*, 381–402. [CrossRef]
33. Boyd, S.; Vandenberghe, L. *Convex Optimization*; Cambridge University Press: Cambridge, UK, 2004.
34. Chen, B.; Zhao, S.; Zhu, P.; Príncipe, J.C. Quantized kernel least mean square algorithm. *IEEE Trans. Neural Netw. Learn. Sys.* **2012**, *23*, 22–32. [CrossRef]

35. Wang, S.; Zheng, Y.; Duan, S.; Wang, L.; Tan, H. Quantized kernel maximum correntropy and its mean square convergence analysis. *Digital Signal Process. Rev. J.* **2017**, *63*, 164–176. [CrossRef]
36. Gil-Alana, L. Long memory behaviour in the daily maximum and minimum temperatures in Melbourne, Australia. *Meteorol. Appl.* **2004**, *11*, 319–328. [CrossRef]
37. FireEye Inc. Out of Pocket: A Comprehensive Mobile Threat Assessment of 7 Million iOS and Android Apps. Available online: https://www.itsecurity-xpert.com/whitepapers/1043-out-of-pocket-a-comprehensive-mobile-threat-assessment-of-7-million-ios-and-android-apps (accessed on 10 July 2019).
38. Ma, Z.; Ge, H.; Liu, Y.; Zhao, M.; Ma, J. A combination method for android malware detection based on control flow graphs and machine learning algorithms. *IEEE Access* **2019**, *7*, 21235–21245. [CrossRef]
39. Makandar, A.; Patrot, A. Malware analysis and classification using artificial neural network. In Proceedings of the International Conference on Trends in Automation, Communication and Computing Technologies, Bangalore, India, 21–22 December 2015; p. 7492653.
40. Miao, Q.; Liu, J.; Cao, Y.; Song, J. Malware detection using bilayer behavior abstraction and improved one-class support vector machines. *Int. J. Inf. Secur.* **2016**, *15*, 361–379. [CrossRef]
41. Tobiyama, S.; Yamaguchi, Y.; Shimada, H.; Ikuse, T.; Yagi, T. Malware detection with deep neural network using process behavior. In Proceedings of the IEEE 40th Annual Computer Software and Applications Conference Workshops, Atlanta, GA, USA, 10–14 June 2016; pp. 577–582.
42. Xu, W.; Qi, Y.; Evans, D. Automatically evading classifiers. In Proceedings of the Network and Distributed Systems Symposium, San Diego, CA, USA, 21–24 February 2016; pp. 21–24.
43. Biggio, B.; Nelson, B.; Laskov, P. Poisoning attacks against support vector machines. In Proceedings of the 29th International Coference on Machine Learning, Edinburgh, Scotland, 26 June–1 July 2012; pp. 1467–1474.
44. Sami, A.; Yadegari, B.; Rahimi, H.; Peiravian, N.; Hashemi, S.; Hamze, A. Malware detection based on mining API calls. In Proceedings of the ACM Symposium on Applied Computing, Sierre, Switzerland, 22–26 March 2010; pp. 1020–1025.
45. Qiao, Y.; Yang, Y.; Ji, L.; He, J. Analyzing malware by abstracting the frequent itemsets in API call sequences. In Proceedings of the 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, Melbourne, VIC, Australia, 16–18 July 2013; pp. 265–270.