This algorithm is based off the Sequential Minimal Optimization algorithm described in:
http://research.microsoft.com/pubs/69644/tr-98-14.pdf

We only need to use vectors that have a Lagrange multiplier > 0 as support vectors, so this algorithm will return the array of multipliers α. An outside algorithm will check if they are > 0 and if so, use those in calculating the weight of the vectors, ignoring the others.

Algorithm 1 Sequential Minimal Optimization SVM

```
procedure smo_svm(input[][], output[], tolerance, c, ϵ)
INPUTS: input - SET OF N INPUT IMAGE VECTORS
        output - SET OF N OUTPUT DECISIONS (-1 OR 1)
        tolerance - CONVERGENCE TOLERANCE
        c - SOFT MARGIN PARAMETER
        ϵ - EPSILON INSENSITIVITY ZONE
    α ← ∅, bias ← 0, error ← 0
    num_changed ← 0, examine ← 1
    while (num_changed > 0 || examine) do
        num_changed ← 0
        if (examine) then
            for (i = 0 → input.size) do
                num_changed+ = examine_example(i)
            end for
        else
            for (i = 0 → input.size) do
                if (α[i] ≠ 0 && α[i] ≠ c) then
                    num_changed+ = examine_example(i)
                end if
            end for
        end if
        if (examine == 1) then
            examine ← 0
        else if (num_changed == 0) then
            num_changed ← 1
        end if
    end while
    return α
end procedure
```

Algorithm 2 examine_example()

```
procedure examine_example(i_2)
INPUTS: i_2 - INDEX TO EXAMPLE IN TRAINING SET
    p_2 ← input[i_2], y_2 ← output[i_2], α_2 ← α[i_2]
    if (α_2 > 0 && α_2 < c) then
        compute_error_2 ← error[i_2]
    else
        compute_error_2 ← compute_svm(p_2) − y_2
    end if
    r_2 ← compute_error_2 ∗ y_2
    #Check Lagrangian multiplier outside of allowable bounds.
    #This tests KKT conditions for 1st choice lagrangian
    if ((r_2 < −tolerance && α_2 < c) && (r_2 > tolerance && α_2 > 0)) then
        #Choose 2nd Lagrangian multiplier to maximize size of step taken during joint optimization.
        #Approximate step size by: |E_1 − E_2|
        i_1 ← 0, max ← 0
        for (i = 0 → input.size) do
            if (α[i] > 0 && α[i] < c) then
                aux_error ← error[i]
                step_size ← |error[i] − compute_error_2|
                if (step_size > max) then
                    max ← step_size
                    i_1 ← i
                end if
            end if
        end for
        if (i_1 ≥ 0 && (take_step(i_1, i_2)) == true) then
            return 1
        end if
        #Sometimes SMO cannot make positive progress with heuristic above. So, iterate through
        #non-bound examples, searching for second example that makes positive progress
        rnd ← random(input.size)
        for (i_1 = rnd → input.size) do
            if (α[i_1] > 0 && α[i_1] < c) then
                if (take_step(i_1, i_2)) == true) then
                    return 1
                end if
            end if
        end for
        for (i_1 = 0 → rnd) do
            if (α[i_1] > 0 && α[i_1] < c) then
                if (take_step(i_1, i_2)) == true) then
                    return 1
                end if
            end if
        end for
```

(Continued on next page)

(Continued from previous page)

```
        #If none of the non-bound examples lead to positive progress, SMO iterates through entire
        #training set until positive progress. If this still doesn't work, abandon this example and move
        #to next iteration for a better one
        rnd ← random(input.size)
        for (i_1 = rnd → input.size) do
            if (take_step(i_1, i_2)) == true) then
                return 1
            end if
        end for
        for (i_1 = 0 → rnd) do
            if (take_step(i_1, i_2)) == true) then
                return 1
            end if
        end for
    else
        return 0
    end if
end procedure
```

Algorithm 3 take_step()

procedure $take\_step(i_1, i_2)$
INPUTS: i_1 - 1ST INDEX TO EXAMPLE IN TRAINING SET
i_2 - 2ND INDEX TO EXAMPLE IN TRAINING SET
  if $(i_1 == i_2)$ **then**
    return false
  end if
  $p_1 \leftarrow input[i_1], y_1 \leftarrow output[i_1], \alpha_1 \leftarrow \alpha[i_1]$
  if $(\alpha_1 > 0 \,\&\&\, \alpha_1 < c)$ **then**
    $compute\_error_1 \leftarrow error[i_1]$
  else
    $compute\_error_1 \leftarrow$ ***compute_svm(*** p_1 ***)*** $- y_1$
  end if
  $p_2 \leftarrow input[i_2], y_2 \leftarrow output[i_2], \alpha_2 \leftarrow \alpha[i_2]$
  if $(\alpha_2 > 0 \,\&\&\, \alpha_2 < c)$ **then**
    $compute\_error_2 \leftarrow error[i_2]$
  else
    $compute\_error_2 \leftarrow$ ***compute_svm(*** p_2 ***)*** $- y_2$
  end if
  $s \leftarrow y_1 * y_2$
  if $(y_1 \neq y_2)$ **then**
    $L \leftarrow max(0, \alpha_2 - \alpha_1)$
    $H \leftarrow min(c, c + \alpha_2 - \alpha_1)$
  else
    $L \leftarrow max(0, \alpha_2 + \alpha_1 - c)$
    $H \leftarrow min(c, \alpha_2 + \alpha_1)$
  end if
  if $(L == H)$ **then**
    return false
  end if
  $k_{11} \leftarrow$ ***gaussian_kernel(*** p_1, p_1 ***)***
  $k_{12} \leftarrow$ ***gaussian_kernel(*** p_1, p_2 ***)***
  $k_{22} \leftarrow$ ***gaussian_kernel(*** p_2, p_2 ***)***
  $\eta \leftarrow k_{11} + k_{22} - 2 * k_{12}$
  if $(\eta > 0)$ **then**
    $a_2 \leftarrow \alpha_2 + y_2 * \frac{compute\_error_1 - compute\_error_2}{\eta}$
    if $(a_2 < L)$ **then**
      $a_2 \leftarrow L$
    end if
    if $(a_2 > H)$ **then**
      $a_2 \leftarrow H$
    end if
  else
    $f_1 \leftarrow y_1 * (compute\_error_1 + bias) - \alpha_1 * k_{11} - s * \alpha_2 * k_{12}$
    $f_2 \leftarrow y_2 * (compute\_error_2 + bias) - s * \alpha_1 * k_{12} - \alpha_2 * k_{22}$
    $L_1 \leftarrow \alpha_1 + s * (\alpha_2 - L)$
    $H_1 \leftarrow \alpha_1 + s * (\alpha_2 - H)$
    $\Psi_L \leftarrow L_1 * f_1 + L * f_2 + \frac{1}{2} * L_1^2 * k_{11} + \frac{1}{2} * L^2 * k_{22} + s * L * L_1 * k_{12}$
    $\Psi_H \leftarrow H_1 * f_1 + H * f_2 + \frac{1}{2} * H_1^2 * k_{11} + \frac{1}{2} * H^2 * k_{22} + s * H * H_1 * k_{12}$

(Continued on next page)

(Continued from previous page)

  if $(\Psi_L < \Psi_H - \epsilon)$ **then**
    $a_2 \leftarrow L$
  else if $(\Psi_L > \Psi_H + \epsilon)$ **then**
    $a_2 \leftarrow H$
  else
    $a_2 \leftarrow \alpha_2$
  end if
end if
if $(|a_2 - \alpha_2| < \epsilon * (a_2 + \alpha_2 + \epsilon))$ **then**
  return false
end if
$a_1 \leftarrow \alpha_1 + s * (\alpha_2 - a_2)$
if $(a_1 < 0)$ **then**
  $a_2 += s * a_1$
  $a_1 \leftarrow 0$
else if $(a_1 > c)$ **then**
  $a_2 += s * (a_1 - c)$
  $a_1 \leftarrow c$
end if
#Update bias
$new\_bias \leftarrow 0, delta\_bias \leftarrow 0$
if $(a_1 > 0\ \&\&\ a_1 < c)$ **then**
  $new\_bias \leftarrow compute\_error_1 + y_1 * (a_1 - \alpha_1) * k_{12} + y_2 * (a_2 - \alpha_2) * k_{22} + bias$
else
  #Both new Lagrangians at bound, choose threshold halfway between them
  $b_1 \leftarrow compute\_error_1 + y_1 * (a_1 - \alpha_1) * k_{11} + y_2 * (a_2 - \alpha_2) * k_{12} + bias$
  $b_2 \leftarrow compute\_error_2 + y_1 * (a_1 - \alpha_1) * k_{12} + y_2 * (a_2 - \alpha_2) * k_{22} + bias$
  $new\_bias \leftarrow \frac{b_1 + b_2}{2}$
end if
$delta\_bias \leftarrow new\_bias - bias$
$bias \leftarrow new\_bias$
#Update error cache
for $(i = 0 \rightarrow input.size)$ **do**
  if $(0 < \alpha[i]\ \&\&\ \alpha[i] < c)$ **then**
    $error[i] += y_1 * (a_1 - \alpha_1) *$ ***gaussian_kernel(***$p_1, input[i]$***)***
      $+ y_2 * (a_2 - \alpha_2) *$ ***gaussian_kernel(***$p_2, input[i]$***)***
      $- delta\_bias$
  end if
end for
$error[i_1] \leftarrow 0$
$error[i_2] \leftarrow 0$
#Update Lagrangians
$\alpha[i_1] \leftarrow a_1$
$\alpha[i_2] \leftarrow a_2$
return true
end procedure

Algorithm 4 compute_svm()

```
procedure compute_svm(p_1)
INPUTS: p_1 - POINT TO COMPUTE DISTANCE AGAINST
    sum ← −bias
    for (i = 0 → input.size) do
        if (α[i] > 0) then
            sum+ = α[i] ∗ output[i] ∗ gaussian_kernel(input[i], p_1)
        end if
    end for
    return sum
end procedure
```

Algorithm 5 guassian_kernel()

```
procedure gaussian_kernel(input[i], p_1)
INPUTS: input[i] - 1ST POINT
        i_1 - 2ND POINT
    guassian ← exp(−(|p_1 − p_2|)^2 / (2∗σ^2))
    return guassian
end procedure
```