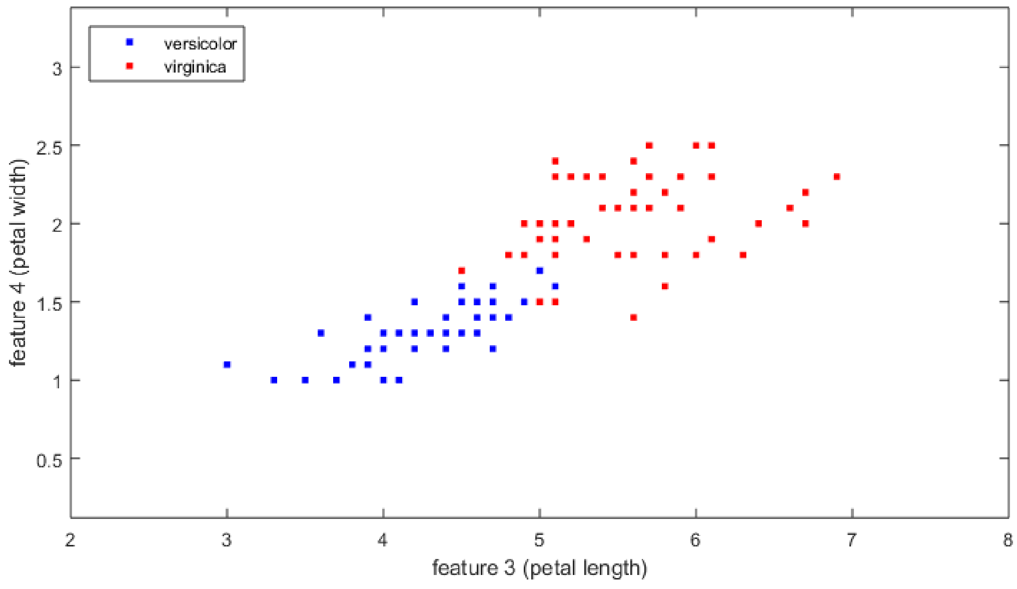
CS/ECE/ME 532
Homework 8: SVM and kernels

due: Monday December 5, 2016

1. Classification and the SVM. Revisit the `iris` data set from Homework 4. For this problem, we will use the 3rd and 4th features to classify whether an iris is *versicolor* or *virginica*. Here is a plot of the data set for this restricted set of features.



We will look for a linear classifier of the form: $x_{i3}w_1 + x_{i4}w_2 + w_3 \approx y_i$. Here, x_{ij} is the measurement of the j^{th} feature of the i^{th} iris, and w_1, w_2, w_3 are the weights we would like to find. The y_i are the labels; e.g. $+1$ for *versicolor* and -1 for *virginica*.

a) Reproduce the plot above, and also plot the decision boundary for the least squares classifier.

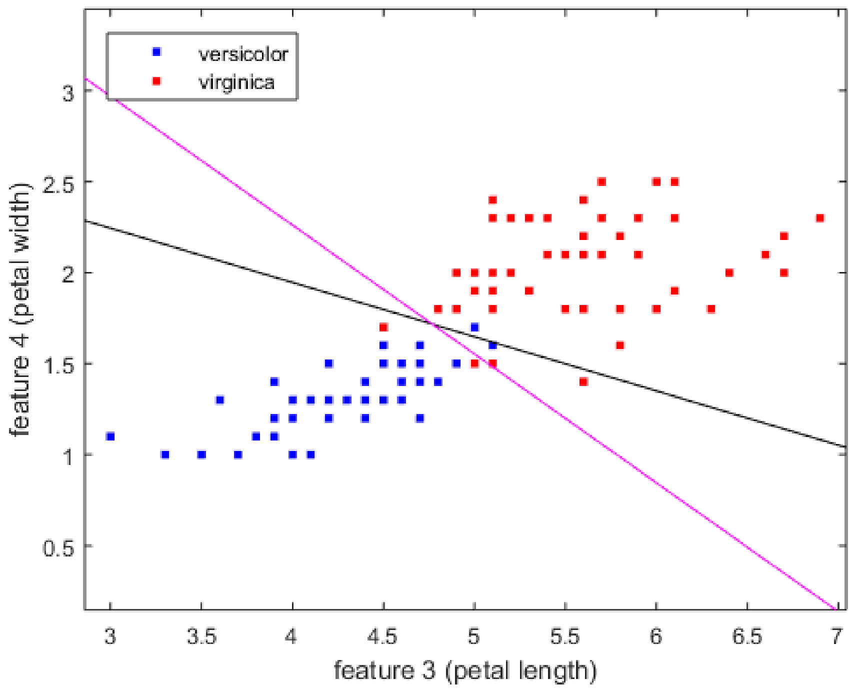
SOLUTION: See solution of part (b).

b) This time, we will use a regularized SVM classifier with the following loss function:

$$\underset{\boldsymbol{w}}{\text{minimize}} \quad \sum_{i=1}^{m} (1 - y_i \boldsymbol{x}_i^\mathsf{T} \boldsymbol{w})_+ + \lambda(w_1^2 + w_2^2)$$

Here, we are using the standard hinge loss, but with an ℓ_2 regularization that penalizes only w_1 and w_2 (we do not penalize the offset term w_3). Solve the problem by implementing gradient descent of the form $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \gamma \nabla f(\boldsymbol{w}_t)$. For your numerical simulation, use parameters $\lambda = 0.1$, $\gamma = 0.003$, $\boldsymbol{w}_0 = \boldsymbol{0}$ and $T = 20{,}000$ iterations. Plot the decision boundary for this SVM classifier. How does it compare to the least squares classifier?

SOLUTION: Here is the plot of the data, the least-squares classifier (black) and the SVM classifiers (magenta).



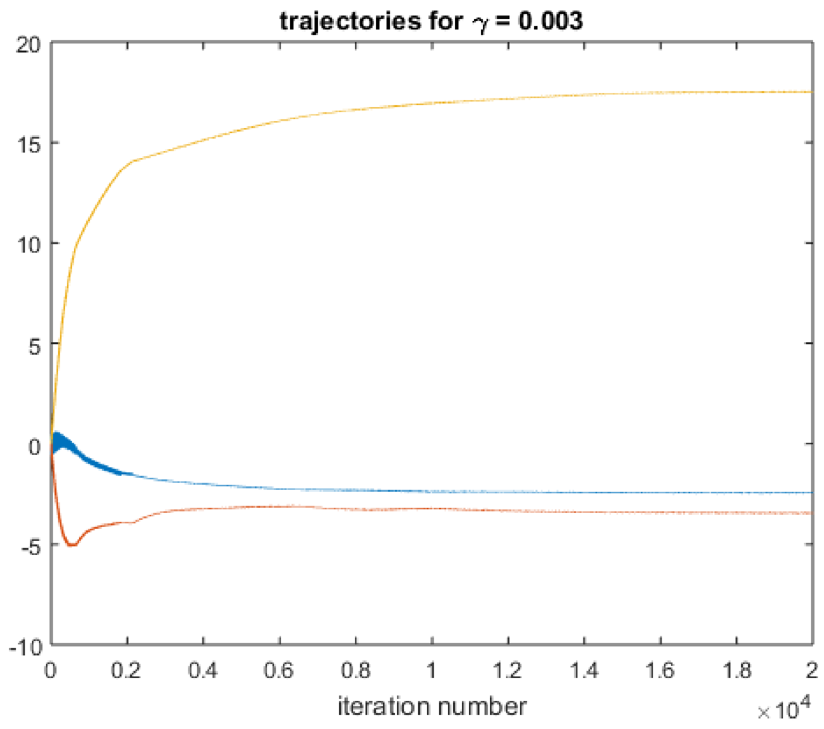
We can observe that both classifiers seem reasonable, though the SVM classifier does a better job (at least visually) of separating the data, and also achieves a slightly better classification error (4 incorrect classifications as opposed to 5 incorrect classifications for the LS classifier). Here is the code that produced these plots:

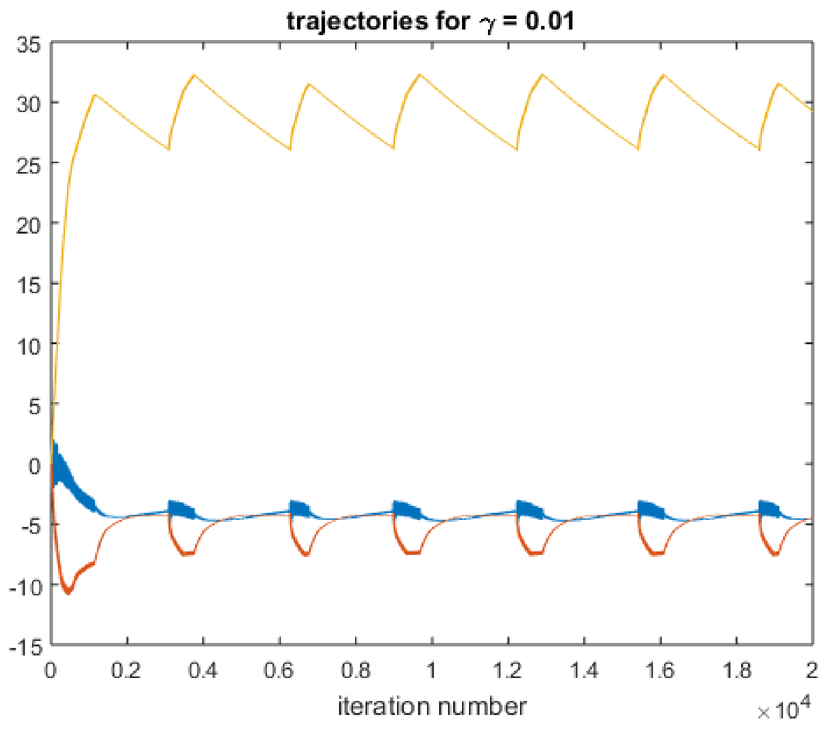
```
load fisheriris
inds = ~strcmp(species,'setosa');
A = meas(inds,[3 4]);
A = [A ones(size(A,1),1)];
y = species(inds);
b = zeros(size(y));

% get blues
figure(1); clf;
ib = find(strcmp(y,'versicolor'));
plot( A(ib,1), A(ib,2), 'b.', 'MarkerSize', 10 )
b(ib) = 1;

% get reds
ir = find(strcmp(y,'virginica'));
b(ir) = -1;

hold on
plot( A(ir,1), A(ir,2), 'r.', 'MarkerSize', 10 )
ax = axis;

legend('versicolor','virginica','Location','northwest')
xlabel('feature 3 (petal length)')
ylabel('feature 4 (petal width)')
axis equal; hold on;

%% solve using LS classifier
```

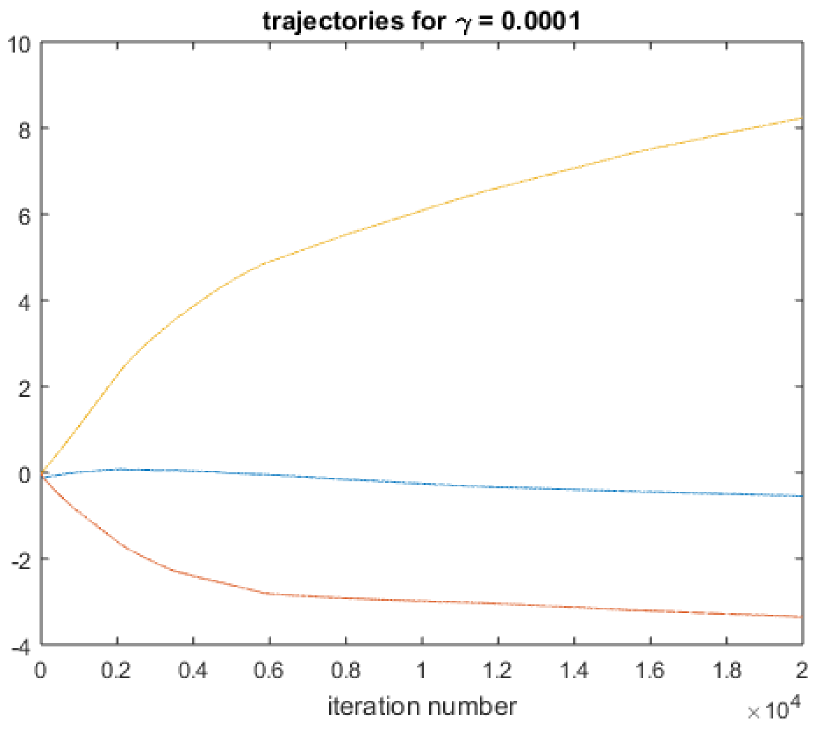
```
what = A\b;
plot( [0 -what(3)/what(1)], [-what(3)/what(2) 0], 'k-' );

% solve using SVM
w = [0;0;0];

lambda = .1;
N = 2e4;
tau = 0.003;

m = size(A,1);
wr = zeros(3,N);
fv = zeros(1,N);
for i = 1:N
    if ~mod(i,1e3)   % display iterations progress
        disp(i)
    end
    wr(:,i) = w;

    dir = 2*lambda*diag([1,1,0])*w;      % find descent direction
    for j = 1:m
        if b(j)*A(j,:)*w < 1
            dir = dir - b(j)*A(j,:)';
        end
    end
    fv(i) = sum(max(0,1-b.*(A*w)));
    w = w - tau*dir;
end

%% figs
plot( [0 -w(3)/w(1)], [-w(3)/w(2) 0], 'm-' );
figure(2);clf; plot(wr');   % plot trajectories as well
title('trajectories for \gamma = 0.003'); xlabel('iteration number')
```

c) Let's take a closer look at the convergence properties of $\boldsymbol{w}_t$. Plot the three components of $\boldsymbol{w}_t$ on the same axes, as a function of the iteration number t. Do the three curves each appear to be converging? Now produce the same plots with a larger stepsize ($\gamma = 0.01$) and a smaller stepsize ($\gamma = 0.0001$). What do you observe?

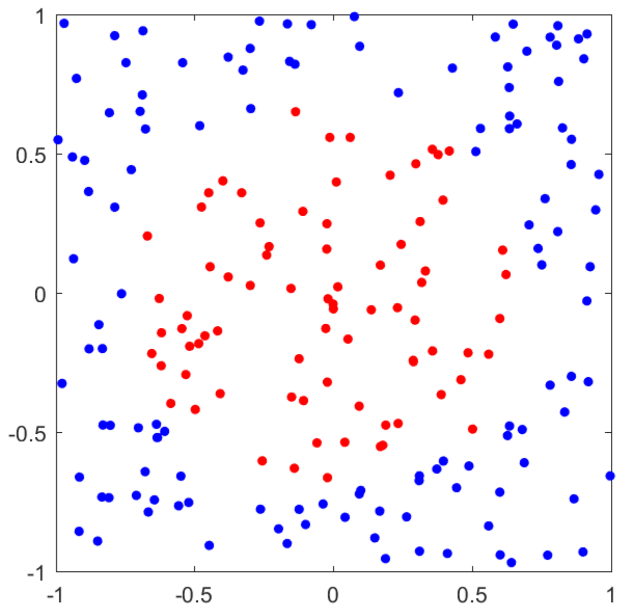
SOLUTION: See below for the three plots. We observe that the trajectories converge and settle nicely for $\gamma = 0.003$. When $\gamma = 0.01$, the trajectories oscillate (stepsize is too big). When $\gamma = 0.0001$, the trajectories still appear to be converging, but just much more slowly (stepsize can be increased).


trajectories for γ = 0.003
20
15
10
5
0
-5
-10
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
iteration number
$\times 10^4$


trajectories for γ = 0.01
35
30
25
20
15
10
5
0
-5
-10
-15
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
iteration number
$\times 10^4$


trajectories for γ = 0.0001
10
8
6
4
2
0
-2
-4
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
iteration number
$\times 10^4$

2. Classification with kernels. Consider the two-dimensional classification problem based on the training data shown in the plot below. The data is provided in the file `circledata.csv`, where the first two columns are the x and y coordinates and the third column is the label ± 1.



a) Solve the standard least squares (linear) classification problem with ℓ_2 regularization parameter $\lambda = 10^{-5}$. Also solve the dual formulation and verify that both produce the same solution.

SOLUTION: The standard regularized least-squares problem is (primal problem):

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \lambda \|\boldsymbol{x}\|_2^2$$

To find the dual, we let $\boldsymbol{x} = \boldsymbol{A}^\mathsf{T}\boldsymbol{\alpha}$ and $\boldsymbol{K} = \boldsymbol{A}\boldsymbol{A}^\mathsf{T}$. Then the dual problem is:

$$\underset{\boldsymbol{\alpha}}{\text{minimize}} \quad \|\boldsymbol{K}\boldsymbol{\alpha} - \boldsymbol{b}\|_2^2 + \lambda\, \boldsymbol{\alpha}^\mathsf{T}\boldsymbol{K}\boldsymbol{\alpha}$$

The primal problem has solution $\widehat{\boldsymbol{x}} = (\boldsymbol{A}^\mathsf{T}\boldsymbol{A} + \lambda\boldsymbol{I})^{-1}\boldsymbol{A}^\mathsf{T}\boldsymbol{b}$. The dual problem solution satisfies the normal equations $\boldsymbol{K}(\boldsymbol{K}+\lambda\boldsymbol{I})\widehat{\boldsymbol{\alpha}} = \boldsymbol{K}\boldsymbol{b}$. Assuming $\boldsymbol{K} \succ 0$ (positive definite), then $\boldsymbol{K}$ is invertible, and the normal equations are equivalent to $(\boldsymbol{K} + \lambda\boldsymbol{I})\widehat{\boldsymbol{\alpha}} = \boldsymbol{b}$. Solving yields $\widehat{\boldsymbol{\alpha}} = (\boldsymbol{K} + \lambda\boldsymbol{I})^{-1}\boldsymbol{b}$. Converting back to the original coordinates, we find:

$$\widehat{\boldsymbol{x}} = \boldsymbol{A}^\mathsf{T}(\boldsymbol{A}\boldsymbol{A}^\mathsf{T} + \lambda\boldsymbol{I})^{-1}\boldsymbol{b} = (\boldsymbol{A}^\mathsf{T}\boldsymbol{A} + \lambda\boldsymbol{I})^{-1}\boldsymbol{A}^\mathsf{T}\boldsymbol{b}$$

and this agrees with our original calculation of $\widehat{\boldsymbol{x}}$ from solving the primal problem.

b) Design a least squares classifier using the Gaussian kernel

$$k(\boldsymbol{a}_i, \boldsymbol{a}_j) = \exp\left(-\tfrac{1}{2}\|\boldsymbol{a}_i - \boldsymbol{a}_j\|^2\right)$$

and regularization parameter $\lambda = 10^{-5}$. Compare its classification of the training data to the least squares classification.

SOLUTION: See solution of part c.

c) Design a least squares classifier using the polynomial kernel

$$k(\boldsymbol{a}_i, \boldsymbol{a}_j) = (\boldsymbol{a}_i^T \boldsymbol{a}_j + 1)^2$$

and regularization parameter $\lambda = 10^{-5}$. Compare its classification of the training data to the least squares classification and Gaussian kernel classifier.

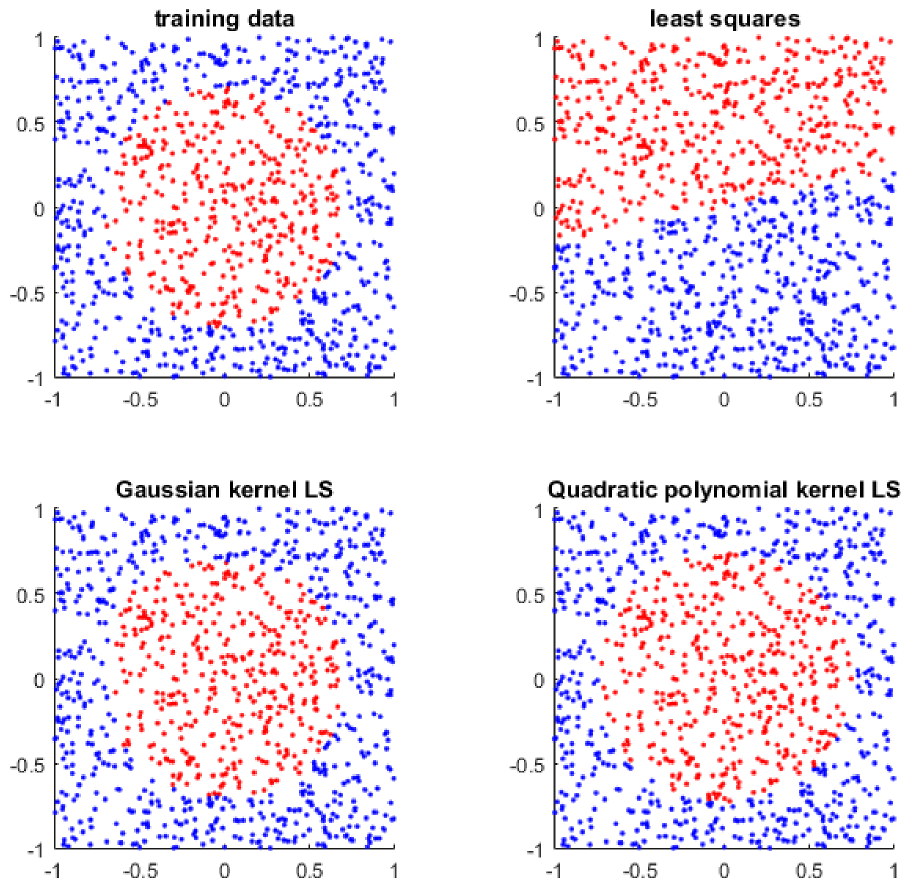
SOLUTION: Instead of using $\boldsymbol{K} = \boldsymbol{A}\boldsymbol{A}^\mathsf{T}$ (i.e. $\boldsymbol{K}_{ij} = a_i^\mathsf{T} a_j$), we use the Gaussian Kernel given by $\boldsymbol{K}_{ij} = \exp\left(-\frac{1}{2}\|a_i - a_j\|^2\right)$ or the polynomial (quadratic) kernel given by $\boldsymbol{K}_{ij} = (a_i^\mathsf{T} a_j + 1)^2$. Implementing these solutions amounts to replacing the original least squares estimator:

```
% least squares (primal formulation with no regularization)
xLS = pinv(A)*b;
bLS = sign(A*xLS);
```

with the kernalized version. Here are the kernalized implementations of LS, Gaussian kernel LS, and polynomial kernel LS:

```
% Regularized least squares (dual formulation)
K = zeros(m);
for i = 1:m
   for j = 1:m
      K(i,j) = A(i,:)*A(j,:)';
   end
end
alphaGLS = pinv(K + lambda*eye(m))*b;
bGLS = sign(K*alphaGLS);

% Gaussian kernel least-squares
K = zeros(m);
for i = 1:m
   for j = 1:m
      K(i,j) = exp( -1/2*norm(A(i,:)-A(j,:))^2 );
   end
end
alphaGLS = pinv(K + lambda*eye(m))*b;
bGLS = sign(K*alphaGLS);

% polynomial kernel (quadratic) least-squares
K = zeros(m);
for i = 1:m
   for j = 1:m
      K(i,j) = ( A(i,:)*A(j,:)' + 1 )^2;
   end
end
alphaGLS = pinv(K + lambda*eye(m))*b;
bGLS = sign(K*alphaGLS);
```

See the next page for a plot of the solution.



The Gaussian versus Polynomial kernels look very similar but they are not quite identical.

d) Now tackle the problem using hinge loss instead of squared error loss. You may do this using the Matlab function `svmtrain`, another package, or by writing your own code (e.g., GD or SGD). Design SVM classifiers using both Gaussian and polynomial kernels and compare your results to those obtained using least squares.

SOLUTION: Implementations will vary depending on which package is used. The Matlab `svmtrain` is straightforward to use. For example, here is code to train the SVM and then see how it performs on the training data.

```
   svmstruct = svmtrain(A,b,'kernel_function','linear');
   bhat = svmclassify(svmstruct,A);
```

To use a Gaussian kernel or a quadratic kernel, simply replace `'linear'` by `'rbf'` or `'quadratic'`, respectively. (`rbf` stands for "Gaussian radial basis function"). The result of the experiment is given in the table below.

Method used	error rate
Linear SVM	0.4097
Gaussian kernel SVM	0.0331
Polynomial kernel SVM	0.0276

The standard linear SVM performs poorly, as expected. Both the Gaussian