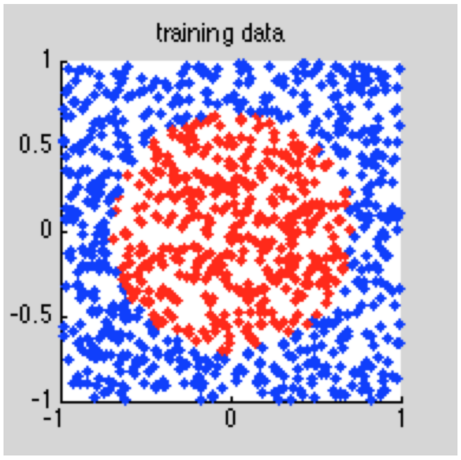
CS/ECE/ME 532
Homework 8: Hinge Loss and SVMs

Consider the classification problem discussed in class, where the goal is to design a classifier based on the training data shown in the plot below. The Matlab file `training_data.m` generates this dataset and also computes the standard least squares solution.



1. Express the standard least squares solution using the dual form using regularization parameter $\lambda = 10^{-5}$. Verify that it generates the same results as the primal 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.

2. Design a classifier using the Gaussian kernel

$$k(\boldsymbol{a}_i, \boldsymbol{a}_j) = \exp(-\frac{1}{2}\|\boldsymbol{a}_i - \boldsymbol{a}_j\|_2^2)$$

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

SOLUTION: See solution of problem 3.

3. Design a 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);
```

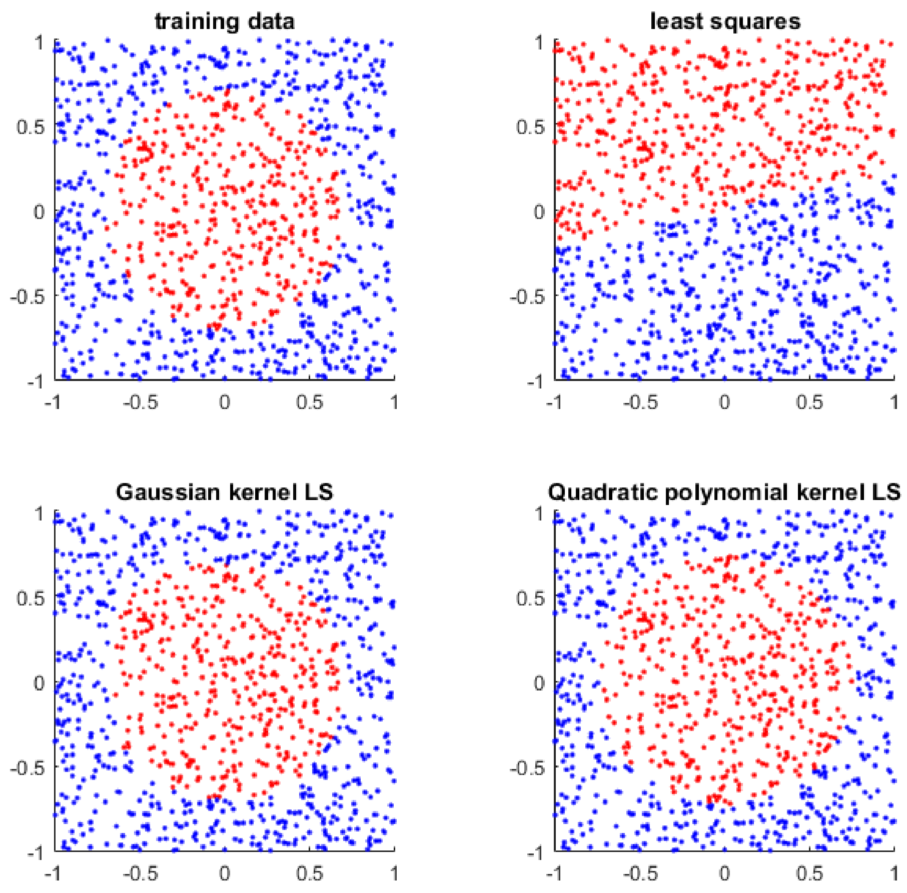
with the kernalized version. Here are the kernalized implementations of LS, Gaussian kernel LS, and polynomial kernel LS:

```
% 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.

4. Now experiment with these methods in the following way. Generate different sized sets of training data: $m = 10, 100, 1000$. Design the three types of classifiers using these data. Test the performance of the classifiers using independent sets of data of size 100. For each training data size, repeat the training and testing 100 times and average the results. This will provide a good indication of how the different approaches perform with different amounts of training data. Summarize your results by reporting the average test error for each of the three methods and each of the three training set sizes.

SOLUTION: Here is a table showing the results of the test. training data size is 10, 100, 1000, and for each size, error rate was averaged over 100 trials.

Method used	$m = 10$	$m = 100$	$m = 1000$
Least-squares	0.3300	0.4462	0.4839
Gaussian kernel LS	0	0.0116	0.0128
Polynomial kernel LS	0.0120	0.0376	0.0375

Ordinary least-squares performs poorly. Slightly better when m is small (perhaps predictably), but as m gets larger, we approach 50% accuracy (no better than guessing). Both kernel methods are much better than ordinary least-squares, and error goes up slightly as we increase the problem size. The Gaussian kernel offered perfect predictive power for the $m = 10$ case (likely overfitting!) and still outperformed the polynomial kernel for large m.

5. Now tackle the problem using hinge loss instead of squared error loss. You do this using the Matlab function `svmtrain`, another package, or by writing your own code (e.g., GD or SGD). Generate different sized sets of training data: $m = 10, 100, 1000$. Design SVM classifiers using both Gaussian and polynomial kernels. Test the performance of the classifiers using independent sets of data of size 100. For each training data size, repeat the training and testing 100 times and average the results. 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	$m = 10$	$m = 100$	$m = 1000$
Linear SVM	0.2960	0.4097	0.4516
Gaussian kernel SVM	0.0770	0.0331	0.0144
Polynomial kernel SVM	0.0400	0.0276	0.0111

The standard linear SVM performs poorly, as expected. The main difference it seems between using kernalized SVM vs kernalized LS is that the kernalized SVM improves in performance as m grows, whereas the kernalized LS gets worse.

6. Consider the problem of trying to predict whether a person is a basketball player based on height. The training data consists of four people with heights of $5'10''$, $5'11''$, $6'1''$ and $6'10''$, and the latter two are basketball players. What classification rule do you obtain by minimizing hinge loss instead of squared error loss?

SOLUTION: The least-squares solution has a decision boundary at $6'2''$ (misclassifies one player), while the linear SVM (with small ℓ_2 regularization to maximize margin) has the decision boundary at $6'0''$, so everyone is classified perfectly.

7. If you consider the dual solution in this case, which values of $\boldsymbol{\alpha} \in \mathbb{R}^4$ are nonzero (i.e., what are the "support vectors")?

SOLUTION: The support vectors are the two players closest to the boundary ($5'11''$ and $6'1''$).

8. The standard hinge loss is linearly decreasing up to 1 and then exactly 0 after. Let t_0 denote the point where the hinge loss first becomes 0. Does the solution change if you shift the t_0 within the range $t_0 \in [0, 1]$?

SOLUTION: The hinge loss we minimize is $\sum_i (1 - y_i \boldsymbol{x}_i^\mathsf{T} \boldsymbol{w})_+$. If the intercept is at t_0 instead of 1, then this amounts to changing the loss function to: $\sum_i (1 - t_0 y_i \boldsymbol{x}_i^\mathsf{T} \boldsymbol{w})_+$. This change does not affect the solution of the problem, since it simply results in $\boldsymbol{w}$ being rescaled.