6.867 Machine learning and neural networks

Problem set 3

Deadline: November 2, 11am in recitation

What and how to turn in?

Turn in short written answers to the questions explicitly stated, and when requested to explain or prove. Do **not** turn in answers when requested to "think", "consider", "try" or "experiment" (except when specifically instructed). You are encouraged to think about, and turn in, answers to questions marked "*optional*"— they will be read and corrected, but a grade will not be recorded for them.

Turn in all MATLAB code explicitly requested, or that you used to calculate requested values. It should be clear exactly what command was used to get the answer to each question.

To help the graders (including yourself...), please be neat, answer the questions briefly, and in the order they are stated. **Staple each "Problem" separately**, and be sure to write your name on the top of every page.

Problem 1: Boosting

This question is about AdaBoost, the simple boosting algorithm presented in the lectures. We assume that there is a method unknown to us that produces a component (weak) hypothesis $h(\mathbf{x})$ in response to a weighted training set. The resulting weak classifier should do at least a bit better than random guessing on the weighted training set it was trained on. We make no assumptions about how well it does on other possibly differently weighted training sets.

Let $D = \{(\mathbf{x}_1, y_1)\ldots,(\mathbf{x}_n, y_n)\}$ be the training set of examples $\mathbf{x}_i$ and binary (±1) labels y_i. We denote the weights on the training examples at the beginning of the k^{th} boosting iteration as $p_k(1),\ldots,p_k(n)$, where $\sum_{i=1}^{n} p_k(i) = 1$. Unlike in the lecture, the component hypotheses $h(\mathbf{x})$ can produce a real valued output (confidence rated classifiers). The sign of the output indicates the label and the magnitude specifies a measure of "confidence" in the classification decision. So, for example, $h(\mathbf{x}) = 10$ would be interpreted as a relatively confident prediction of label 1.

Now, starting with equal weights $p_1(i) = 1/n$, AdaBoost generates a sequence of hypotheses $h_1(\mathbf{x}), \ldots, h_m(\mathbf{x})$, where each $h_k(\mathbf{x})$ was trained with a different set of example weights. After having generated a hypothesis $h_k(\mathbf{x})$ in response to weights $p_k(i)$ at the k^{th} iteration, the example weights are updated according to

$$p_{k+1}(i) = c \cdot p_k(i)\, \exp(\, -\hat{\alpha}_k y_i h_k(\mathbf{x}_i)\,), \quad i = 1, \ldots, n \tag{1}$$

where c is the normalization constant ensuring that $\sum_{i=1}^{n} p_{k+1}(i) = 1$ and $\hat{\alpha}_k$ gives the votes that we assign to the new component classifier $h_k(\mathbf{x})$. We would like to assign these votes so that the new component classifier $h_k(\mathbf{x})$ is at chance performance relative to the new weights $p_{k+1}(i)$. One way to express this is as follows:

$$\sum_{i=1}^{n} p_{k+1}(i) y_i h_k(\mathbf{x}_i) = 0 \tag{2}$$

1. Show that if $h_k(\mathbf{x}_i)$ generates only binary ± 1 outputs, then the above condition corresponds to requiring that the training error is exactly 0.5 relative to the new weights

   Answer: In the sum $\sum_{i=1}^{n} p_{k+1}(i) y_i h_k(x_i)$, let P be the set of i for which $p_{k+1}(i)$ is multiplied by $+1$, and Q be the set of i for $p_{k+1}(i)$ is multiplied by -1. Then since $\sum_{i=1}^{n} p_{k+1}(i) y_i h_k(x_i) = 0$, we have $\sum_{i \in P} p_{k+1}(i) = \sum_{j \in Q} p_{k+1}(j)$. Also, since $\sum_{i=1}^{n} p_{k+1}(i) = 1$, we have $\sum_{i \in P} p_{k+1}(i) + \sum_{j \in Q} p_{k+1}(j) = 1$, so $\sum_{i \in P} p_{k+1}(i) = \frac{1}{2}$. But, $\sum_{i \in P} p_{k+1}(i) = \sum_{h_k(x_i) = y_i} p_{k+1}(i)$, so the training error of h_k relative to p_{k+1} is $\frac{1}{2}$.

Scoring: *4 points*

Things are a bit different here since the component classifier can generate real valued outputs. We can view the condition instead as a way of *decorrelating* the predictions and the labels relative to the new weights.

2. Show that if we choose $\hat{\alpha}_k$ as the minimizing argument of

$$J(\alpha) = \log \left(\sum_{i=1}^{n} p_k(i) \exp\{-\alpha y_i h_k(\mathbf{x}_i)\} \right) \tag{3}$$

   we will indeed ensure that the decorrelation condition (2) holds for the new weights. (Hint: Do not attempt to compute $\hat{\alpha}_k$. Instead, set the derivative to zero and use the resulting equation, which $\hat{\alpha}_k$ must satisfy, in order to show (2) is satisfied.

Answer: Let $\hat{\alpha}_k$ be the minimizing argument of $J(\alpha)$ above, then $J'(\hat{\alpha}_k) = 0$. Taking the derivative of J, we get:

$$\begin{aligned}
0 &= \frac{\partial}{\partial \hat{\alpha}_k} \log\left(\sum_{i=1}^n p_k(i) \exp\{-\hat{\alpha}_k y_i h_k(\mathbf{x}_i)\}\right) \\
&= \frac{\sum_{i=1}^n p_k(i) \frac{\partial}{\partial \alpha} e^{-\hat{\alpha}_k y_i h_k(\mathbf{x}_i)}}{\sum_{i=1}^n p_k(i) e^{-\hat{\alpha}_k y_i h_k(\mathbf{x}_i)}} \\
&= -\frac{\sum_{i=1}^n p_k(i) y_i h_k(\mathbf{x}_i) e^{-\hat{\alpha}_k y_i h_k(\mathbf{x}_i)}}{\sum_{i=1}^n p_k(i) e^{-\hat{\alpha}_k y_i h_k(\mathbf{x}_i)}} \\
&= -\sum_{i=1}^n \frac{p_k(i) e^{-\hat{\alpha}_k y_i h_k(\mathbf{x}_i)}}{\sum_{j=1}^n p_k(j) e^{-\hat{\alpha}_k y_j h_k(\mathbf{x}_j)}} y_i h_k(\mathbf{x}_i)
\end{aligned}$$

Matching this with the definition of the weights $p_{k+1}(i)$ given in (1), and noticing that the denominator is exactly the correct normalization constant $1/c$:

$$= -\sum_{i=1}^n p_{k+1}(i) y_i h_k(\mathbf{x}_i)$$

This is the desired correlation equality.

Scoring: *5 points*

This should look a bit suspicious... as if there were an objective function that the boosting algorithm were minimizing at each iteration. This is indeed the case. Let $H_m(\mathbf{x})$ be the combined hypothesis resulting from m boosting iterations:

$$H_m(\mathbf{x}) = \hat{\alpha}_1 h_1(\mathbf{x}) + \ldots + \hat{\alpha}_m h_m(\mathbf{x}) \tag{4}$$

We'd like you to show that

$$J(H_m) = \log\left(\sum_{i=1}^n \exp\{-y_i H_m(\mathbf{x}_i)\}\right) \tag{5}$$

serves as an objective function for the boosting algorithm. In other words, we claim that every time we add a new component hypothesis to the combined classifier $H_m(\mathbf{x})$, we decrease the objective.

3. Show that $J(H_m) \geq J(H_{m+1})$, where

$$H_{m+1}(\mathbf{x}) = H_m(\mathbf{x}) + \hat{\alpha}_{m+1} h_{m+1}(\mathbf{x}), \tag{6}$$

$h_{m+1}(\mathbf{x})$ is the new component hypothesis, and $\hat{\alpha}_{m+1}$ is optimized as shown above. (Hint: expand the weights $p_m(i)$ and use the optimization of $\hat{\alpha}_{m+1}$).

Answer: Denote by Z_{k+1} the normalization constant $\frac{1}{c}$ in (1):

$$Z_{k+1} = \sum_i p_k(i) e^{-\hat{\alpha}_k y_i h_k(\mathbf{x}_i)}$$

so that (1) can be rewritten as:

$$p_{k+1}(i) = \frac{1}{Z_{k+1}} p_k(i) e^{-\hat{\alpha}_k y_i h_k(\mathbf{x}_i)}.$$

Unrolling this recursive definition of the weights yields:

$$\begin{aligned}
p_{m+1}(i) &= \frac{1}{Z_{m+1}} p_m(i) e^{-\hat{\alpha}_m y_i h_m(\mathbf{x}_i)} \\
&= \frac{1}{Z_{m+1}} \frac{1}{Z_m} p_{m-1}(i) e^{-\hat{\alpha}_{m-1} y_i h_{m-1}(\mathbf{x}_i)} e^{-\hat{\alpha}_m y_i h_m(\mathbf{x}_i)} \\
= \cdots &= \prod_{k=2}^{m+1} \frac{1}{Z_k} p_1(i) \prod_{k=1}^{m} e^{-\hat{\alpha}_k y_i h_k(\mathbf{x}_i)}
\end{aligned}$$

initially $p_1(i) = \frac{1}{n}$, hence:

$$= \frac{e^{-y_i \sum_{k=1}^{m} \hat{\alpha}_k h_k(\mathbf{x}_i)}}{n \prod_{k=2}^{m+1} Z_k}$$

noticing that the sum in the exponent corresponds to the combined classifier H_m:

$$= \frac{e^{-y_i H_m(\mathbf{x}_i)}}{n \prod_{k=2}^{m+1} Z_k}$$

We now note that $\sum_i p_{m+1}(i) = 1$, and thus:

$$\begin{aligned}
0 = \log \sum_i p_{m+1}(i) &= \log \left(\sum_i \frac{e^{-y_i H_m(\mathbf{x}_i)}}{n \prod_{k=2}^{m+1} Z_k} \right) \\
&= \log \left(\sum_i e^{-y_i H_m(\mathbf{x}_i)} \right) - \sum_{k=2}^{m+1} \log Z_k - \log n \\
&= J(H_m) - \sum_{k=2}^{m+1} \log Z_k - \log n
\end{aligned}$$

Noting that $\log Z_{k+1} = J(\hat{\alpha}_k)$ we can rewrite $J(H_m)$ as:

$$J(H_m) = \log n + \sum_{k=1}^{m} J(\hat{\alpha}_k)$$

Consequently, $J(H_{m+1}) = J(H_m) + J(\hat{\alpha}_{m+1})$. Noting that $\hat{\alpha}_{m+1}$ was chosen to minimize J:

$$\begin{aligned}
J(\hat{\alpha}_{m+1}) &\leq J(\alpha) \quad \text{For any } \alpha \\
J(\hat{\alpha}_{m+1}) &\leq J(0) \\
&= \log\left(\sum_{i=1}^{n} p_m(i) e^{-0 y_i h_m(\mathbf{x}_i)}\right) \\
&= \log\left(\sum_{i=1}^{n} p_m(i) e^{0}\right) = \log \sum_{i=1}^{n} p_m(i) \\
&= \log 1 = 0
\end{aligned}$$

Hence $J(H_{m+1}) \leq J(H_m)$.

Scoring: *6 points*

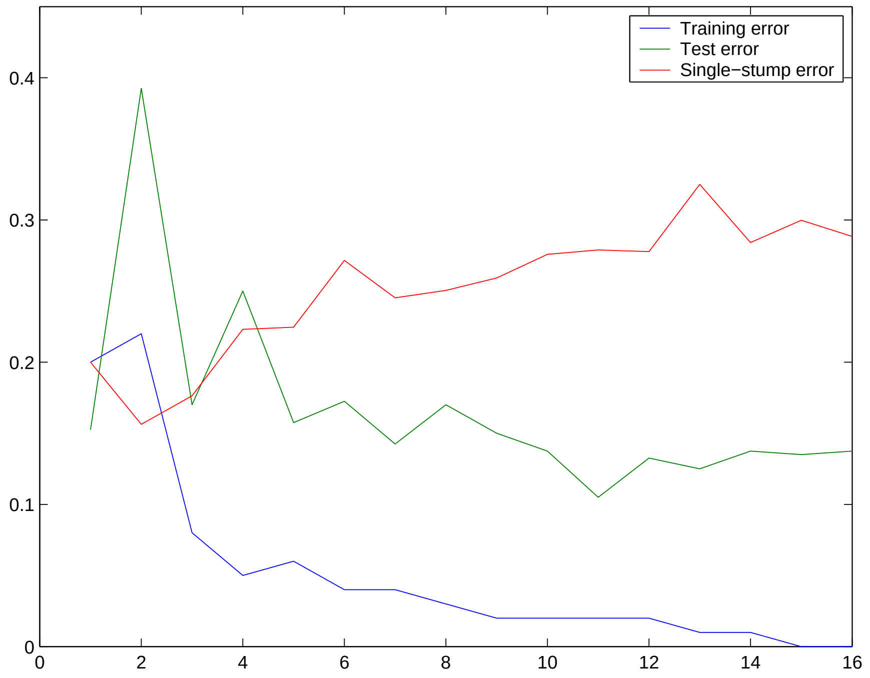
Having now put some effort into understanding the boosting algorithm, let's explore a bit how it behaves in practice. We have provided you with MATLAB code that finds and evaluates (confidence rated) decision stumps. These are the hypothesis that our boosting algorithm assumes we can generate. The relevant MATLAB files are "boost_digit.m", "boost.m", "eval_boost.m", "find_stump.m", "eval_stump.m". You'll only have to make minor modifications to "boost.m" and, a bit later, to "eval_boost.m" and "boost_digit.m" to make these work.

4. Complete the weight update in "boost.m" and run "boost_digit" to plot the training and test errors for the combined classifier as well as the corresponding training error of the decision stump, as a function of the number of iterations. Are the errors what you would expect them to be? Why or why not?

   Answer: The completed `boost.m`:

```
function [param] = boost(X,y,niter)
 n = size(X,1);
 p = ones(n,1)/n;
 param = [];
 for i=1:niter,
   stump = find_stump(X,y,p);
   h = eval_stump(X,stump); % stump predictions on the training set
   %  compute the number of votes to give to the new stump
   epsilon = sum(p .* (y ~= h)) ;
   alpha = 0.5*log((1-epsilon)/epsilon) ;
   %  update the weights
   p = p .* exp(-alpha * y .* h) ;
```

```
  p = p / sum(p) ; % don't forget to normalize !
  param = [param,struct('stump',stump,'alpha',alpha)];
end;
```



As can be seen in the figure, the training and test errors decrease as we perform more boosting iterations. Eventually the training error reaches zero, but we do not overfit, and the test error remains low (though higher than the training error). However, no single stump can predict the training set well, and especially since we continue to emphasize "difficult" parts of the training set, the error of each particular stump remains high, and does not drop bellow about 1/3.

Scoring: *6 points for the execution, 3 points for the explanation*

We will now investigate the classification margins of training examples. Recall that the classification margin of a training point in the boosting context reflects the "confidence" in which the point was classified correctly. You can view the margin of a training example as the difference between the weighted fraction of votes assigned to the correct label and those assigned to the incorrect one. Note that this is not a geometric notion of "margin" but one based on votes. The margin will be positive for correctly classified training points and negative for others.

5. Modify "eval_boost.m" so that it returns normalized predictions (normalized by the total number of votes). The resulting predictions should be in the range $[-1, 1]$. Fill in the missing computation of the training set margins in "boost_digit.m" (that is, the classification margins for each of the training points). You should also uncomment

the plotting script for cumulative margin distributions (what is plotted is, for each $-1 < r < 1$ on the horizontal axis, what fraction of the training points have a margin of at least r). Explain the differences between the cumulative distributions after 4 and 16 boosting iterations.

Answer: The completed eval_boost.m:

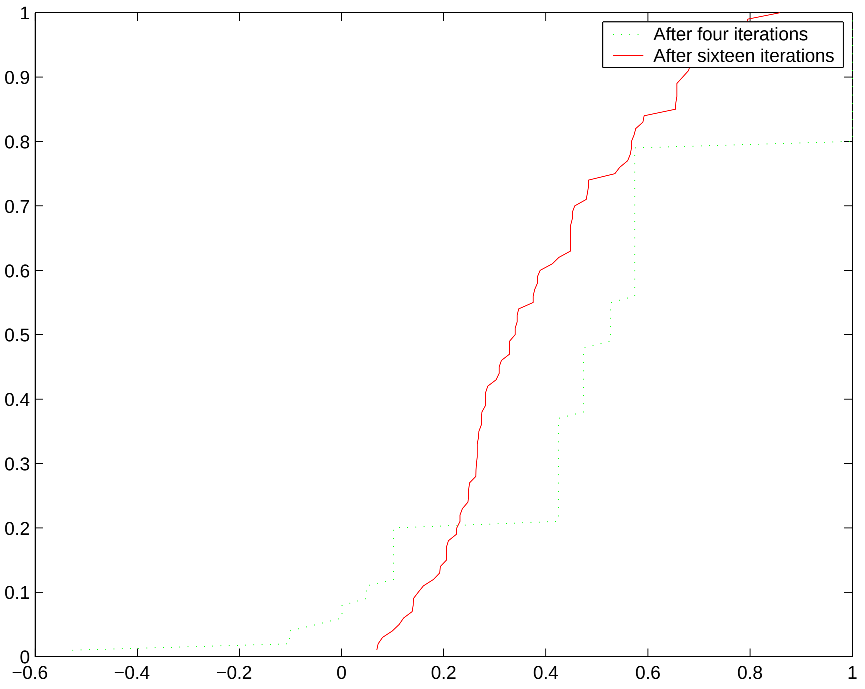
```
function [H] = eval_boost(X,param);
 [n,m] = size(X);
 H = zeros(n,1); % combined predictions
 totalalpha = 0; % Total weight of all votes
 for i = 1:length(param),
   H = H + param(i).alpha*eval_stump(X,param(i).stump);
   totalalpha = totalalpha + param(i).alpha ;
 end;
 H = H/totalalepha ; % Normalize by the total vote weight
```

The completed boost_digit.m:

```
load digit_x.dat;
load digit_y.dat;
digit_y = 2*digit_y-1;

load digit_x_test.dat;
load digit_y_test.dat;
digit_y_test = 2*digit_y_test-1;

E=[];
M=[];
for i=1:16,

  param = boost(digit_x,digit_y,i);
  yb = eval_boost(digit_x,param);
  trainerr= mean(yb.*digit_y<=0);

  % classification margins on the training set
  % mar = a column vector of length length(digit_y), where
  %       mar(i) is the classification margin of the ith training point
  mar = yb .* digit_y ;
  M = [M,mar];

  yb = eval_boost(digit_x_test,param);
  testerr = mean(yb.*digit_y_test<=0);
  E=[E;trainerr,testerr,param(i).stump.err];
```

```
end;

figure(1); plot(E);
legend('Training error','Test error','Single-stump error');

figure(2);
n = length(digit_y);
plot(sort(M(:,4)),(1:n)'/n,':g',sort(M(:,16)),(1:n)'/n,'-r');
legend('After four iterations','After sixteen iterations');
```

(note that we also added code to add legends to the figures).



The key difference between the cumulative distributions after 4 and 16 boosting iterations is that the additional iterations seem to push the left (low end) tail of the cumulative distribution to the right. To understand the effect, note that the examples that are difficult to classify have poor or negative classification margins and therefore define the low end tail of the cumulative distribution. Additional boosting iterations concentrate on the difficult examples and ensure that their margins will improve. As the margins improve, the left tail of the cumulative distribution moves to the right, as we see in the figure.

Scoring: *4 points for the execution, 2 points for the explanation*

Problem 2: support vector machines

Lagrange multipliers and optimization problems

We'll present here a very simple tutorial example of using and understanding Lagrange multipliers. Let w be a scalar parameter we wish to estimate and x a fixed scalar. We wish to solve the following (tiny) SVM like optimization problem:

$$\text{minimize } \frac{1}{2}w^2 \text{ subject to } wx - 1 \geq 0 \tag{7}$$

This is difficult only because of the constraint. We'd rather solve an unconstrained version of the problem but, somehow, we have to take into account the constraint. We can do this by including the constraint itself in the minimization objective as it allows us to twist the solution towards satisfying the constraint. We need to know how much to emphasize the constraint and this is what the Lagrange multiplier is doing. We will denote the Lagrange multiplier by α to be consistent with the SVM problem. So we have now constructed a new minimization problem (still minimizing with respect to w) that includes the constraint as an additional linear term:

$$J(w;\alpha) = \frac{1}{2}w^2 - \alpha(wx - 1) \tag{8}$$

The Lagrange multiplier α appears here as a parameter. You might view this new objective a bit suspiciously since we appear to have lost the information about what type of constraint we had, i.e., whether the constraint was $wx - 1 \geq 0$, $wx - 1 \leq 0$, or $wx - 1 = 0$. How is this information encoded? We can encode this by constraining the values of the Lagrange multipliers:

$$\begin{array}{lll} wx - 1 \geq 0 & \Rightarrow & \alpha \geq 0 \\ wx - 1 \leq 0 & \Rightarrow & \alpha \leq 0 \\ wx - 1 = 0 & \Rightarrow & \alpha \text{ is unconstrained} \end{array}$$

Note, for example, that when the constraint is $wx - 1 \geq 0$, as we have above, large positive values of α will encourage choices of w that result in large positive values for $wx - 1$. This is because in the above objective, $J(w;\alpha)$, we try to minimize $-\alpha(wx - 1)$ in addition to $w^2/2$; minimizing $-\alpha(wx - 1)$ is the same as maximizing $\alpha(wx - 1)$ or $wx - 1$ since α is positive. Figure 1 tries to illustrate this effect. Assuming $x = 1$ we can plot the new objective function as a function of w for different values of α. Larger values of α clearly move the solution (minimizing w) towards satisfying $w - 1 \geq 0$ (we assume here that $x = 1$). Based on the figure we can see that setting $\alpha = 1$ produces just the right solution, i.e., $w^* = 1$, which satisfies the constraint $wx - 1 \geq 0$ (when $x = 1$) with minimal distortion of the original objective. There's no reason to consider negative values for α since they would push the solution away from satisfying our inequality constraint.

Effectively what we are doing here is solving a large number of optimization problems, once for each setting of the Lagrange multiplier α. Indeed, we can express the solution (the

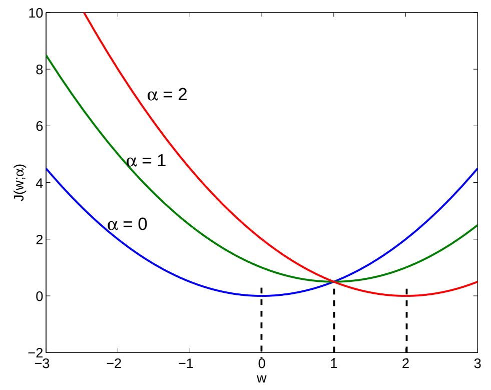


Figure 1: $J(w;\alpha)$ as a function of w for different values of α. The minimizing w values are indicated with dashed line segments. x was set to 1.

minimizing w) as a parametric function of α:

$$\frac{\partial}{\partial w} J(w;\alpha) = w - \alpha x = 0 \tag{9}$$

meaning that $w^*_\alpha = \alpha x$. We could now find the setting of α such that the constraint $w^*_\alpha x - 1 \geq 0$ is satisfied. There are multiple answers to this since larger values of α would better satisfy the constraint. Finding the smallest (non-negative) α for which the constraint is satisfied would in this case produce the right solution (one corresponding to the minimal change of the original problem).

We can proceed a bit more generally, however, the way we handled the quadratic optimization problem for SVMs. Let's insert our solution w^*_α back into the objective function:

$$J(w^*_\alpha;\alpha) \;=\; \frac{1}{2}(w^*_\alpha)^2 - \alpha(w^*_\alpha x - 1) = \frac{1}{2}(\alpha x)^2 - \alpha(\alpha x^2 - 1) = \alpha - \frac{1}{2}(\alpha x)^2 \tag{10}$$

The result, which we denote as $J(\alpha)$, is a function of the Lagrange multiplier α only. Let's understand this function a bit better. In Figure 1, the values of the objective at the dashed lines correspond exactly to $J(w^*_\alpha;\alpha)$ or $J(\alpha)$, evaluated at $\alpha = 0, 1, 2$. Isn't it strange that the right solution ($\alpha = 1$) appears to yield the maximum of $J(\alpha)$? This is a very useful property. Let's verify this by finding the maximum of $J(\alpha)$ a bit more formally:

$$J(\alpha) \;=\; \alpha - \frac{1}{2}(\alpha x)^2 \tag{11}$$

$$\frac{\partial}{\partial \alpha} J(\alpha) \;=\; 1 - \alpha x^2 = 1 - w^*_a x = 0 \tag{12}$$

where we have used our previous result $w^*_\alpha = \alpha x$. So, the constraint is satisfied with equality at the maximum of $J(\alpha)$. More rigorously, since $\alpha \geq 0$ in our setting, the maximum is obtained either at $\alpha = 0$ or at the point where $1 - w^*_a x = 0$. We can express this more concisely by saying that their product vanishes, i.e., $\alpha(w^*_a x - 1) = 0$ at the optimum. This is generally true, i.e., either the Lagrange multiplier is not used and $\alpha = 0$ (the constraint is satisfied without any modification) or the Lagrange multiplier is positive and the constraint is satisfied with equality.

The remaining question for us here is why

$$\text{maximize } \alpha - \frac{1}{2}(\alpha x)^2 \text{ subject to } \alpha \geq 0 \tag{13}$$

is any better than the problem we started with. The short answer is that the constraints here are very simple non-negativity constraints that are easy to deal with in the optimization. In the SVM context, we have another reason to prefer this formulation.

Examples of SVM training

We provided a skeleton for a SVM training routine `trainsvm.m`. The routine takes as input the training set, a constant C which will be an upper bound on the Lagrange multipliers, and a specification for a kernel function. Your first task is to complete this routine. Two parts of the routine are missing.

First, you should set up and solve the quadratic programming problem:

$$\text{minimize } \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) - \sum_i \alpha_i$$

subject to: $\sum_i \alpha_i y_i = 0$ and for all i, $0 \leq \alpha_i \leq C$. Solve the quadratic programming using the MATLAB routine `quadprog`. (Note that in lecture we saw this as a maximization problem. MATLAB's `quadprog` solves quadratic *minimization* problems, and so we changed the sign of the objective function to get an equivalent minimization problem).

Next, add code to calculate the bias term w_0. We have already provided code for finding vectors that lie on the margin (note that we account for numeric inaccuracies by ignoring α values very close to zero or to C). Although it is enough to use one of these input vectors for calculating w_0, in order to increase accuracy, we will calculate w_0 using each one of these input vectors separately, and then take w_0 to be the median of all the calculated values (all of which should theoretically be the same).

Note that the routine also returns a list of the support vectors, and the vectors that lie exactly on the margin.

1. Why do we use only training vectors $\mathbf{x}_i$ with $0 < \alpha_i < C$ for calculating w_0 ?

   Answer: We know what the prediction $\mathbf{w}^T\mathbf{x} + w_0$ should be on points $\mathbf{x}$ which are exactly on the margin, that is with the constraint $y_i(\mathbf{w}^T\mathbf{x} + w_0) \geq 1$ tightly satisfied.

   Points that are loosely satisfied by the constraints, i.e. with $y_i(\mathbf{w}^T\mathbf{x}+w_0) > 1$, will necessarily have their Lagrange multiplier equal to zero (since higher Lagrange multipliers will hurt the objective function).

If we impose some limit C on the Lagrange multipliers, then if a point does not satisfy the constraint, we would like to increase its Lagrange multiplier as high as possible, and so its Lagrange multiplier will be equal to C.

Taking only points with Lagrange multipliers strictly between zero and C ensures that all the points considered are on the margin (note that there might still be margin vectors with Lagrange multipliers equal to zero or C, and we will miss them).

Scoring: *2 points*

2. Complete the routine `trainsvm.m`.

   Answer: Here is the completed routine. Note that we also compute the margin (which is requested in some of the questions).

```
function [a,w0,fval,supvecs,margvecs,margin] = trainsvm(x,y,C,...
                                                kernalfunc,varargin)

  if nargin<4
      kernalfunc = 'polyK';
  end
  if nargin<3
      C = 1e+10; % should be inf. Set like this because of
  end            % bug with quadprog

  n = size(x,1);
  K = feval(kernalfunc,x,x,varargin{:});
  [a,fval,exitflag] = quadprog(diag(y)*K*diag(y),-ones(n,1),...
      [],[],y',0,zeros(n,1),C*ones(n,1));
  ay = a .* y;
  eps = 0.001*max(a);
  supvecs = find(a>=eps);
  margvecs = find(a>=eps & a<=C-eps);
  w0 = median(y(margvecs)-K(margvecs,:)*ay);
  % Calculate the margin in feature space, which is 1/||w||
  margin = 1/sqrt(ay'*K*ay) ;
```

Scoring: *6 points (not including the margin computation)*

3. Write a routine `svmpred.m` that classifies a set of input vectors based on a specified SVM classifier. The routine should take as input a matrix of input vectors, and the specifications of the SVM classifier. You may use the provided skeleton.

   Answer:

```
function [y,z] = svmpred(x,tx,ty,a,w0,kernalfunc,varargin)
  if nargin<6
      kernalfunc = 'polyK';
  end
  z = feval(kernalfunc,x,tx,varargin{:})*(a.*ty)+w0;
  y=sign(z);
```

Scoring: *5 points*

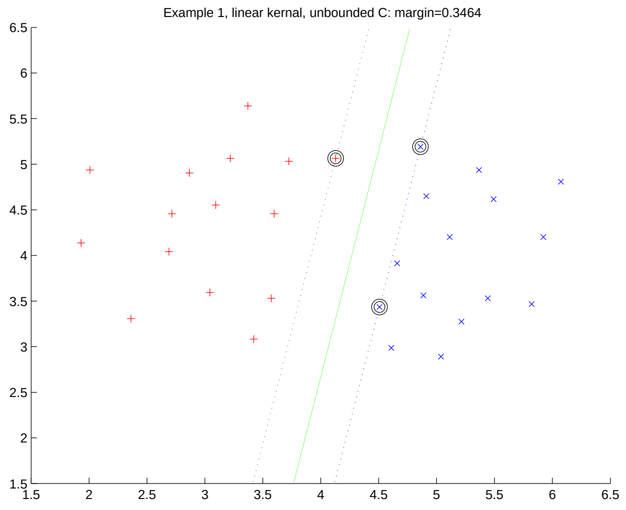
In studying SVM training, it will be useful to plot the decision boundary, the margin, and the support vectors. The provided routine `plotsvm2d.m` does just that. The routine takes as input the training set, the lists of support and margin vectors, the name of the prediction function (which will probably be `svmpred`), and the classifier specifications to be passed to the prediction function. (If you deviated from the provided skeleton of `svmpred.m`, you might need to change `plotsvm2d.m` appropriately).

The example data sets used in this question can be found in the file `svmexample.mat`, and can be read using:

```
>> load svmexample
```

4. We start with a simple example, given in `train1x`, `train1y`. Use the routines above to train a SVM using a linear kernel function $K(\mathbf{x}_1, \mathbf{x}_2) = (1 + \mathbf{x}_1^T \mathbf{x}_2)$ (note the provided `polyK.m`), and without bounding α_i from above. Turn in the plot created by `plotsvm2d.m`.

Answer:



Scoring: *1 point*

5. Why is it not necessary to bound the values of α_i from above in this case ?

Answer: The data set is separable, and so the dual problem is bounded. Increasing α to infinity will not better the objective function of the dual quadratic programming.

Scoring: *2 points*

6. What is the classification margin ? (Explain how you calculated it, providing any relevant MATLAB code)

   Answer: The margin, as computed by `trainsvm.m` above, is 0.3464.

   Scoring: *3 points (including the explanation/code)*

We now move on to a slightly more complex data set given in `train2x`, `train2y`. This time it is necessary to bound the values of α_i by some finite C.

7. In order to understand what happens when we do not impose such a bound, try bounding with increasingly higher constants C and plot the value of the quadratic programming objective function as a function of C (turn in this plot). Why can we not solve the quadratic optimization problem when the values of α_i are unbounded ?

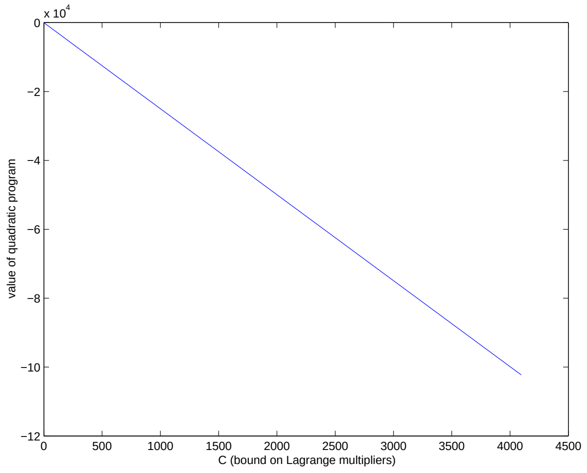
   Answer: Using the following routine:

```
function fvals = calcfvals(x,y,Cs,varargin)
 fvals = zeros(size(Cs));
 for i=1:length(Cs)
  [a,w0,fvals(i),supvecs,margvecs,margin] = trainsvm(x,y,Cs(i),varargin{:});
 end
```

We can generate the plot using:

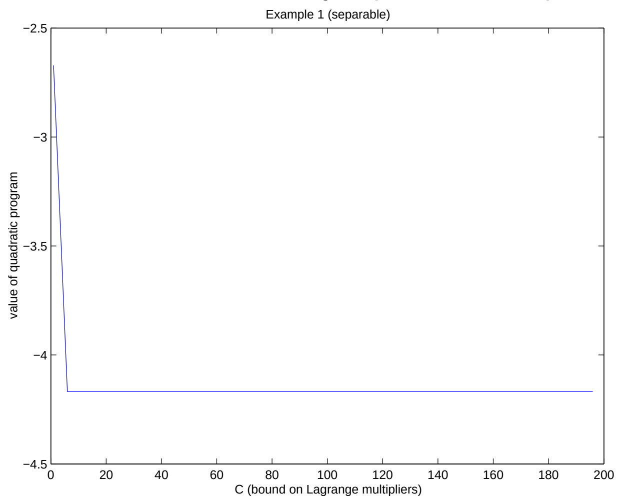
```
>> Cs = 2*(0:12) ;
>> fvals = calcfvals(train2x,train2y,Cs) ;
>> plot(Cs,fvals) ;
```

We get the following plot:



From this plot, we can see that the objective function continues to decreases as the maximum value of the Lagrange multiplier increases. If the Lagrange multiplier are not bounded, the value of the quadratic program is unbounded, and there is no optimal solutions.

For comparison, consider the following plot of the objective function as a function of C, for the linearly separable example one:



In this case, the objective function initially decreases (it is always monotonically non-increasing), but then reaches a lower bound, and does not change further even as the bound C increases.

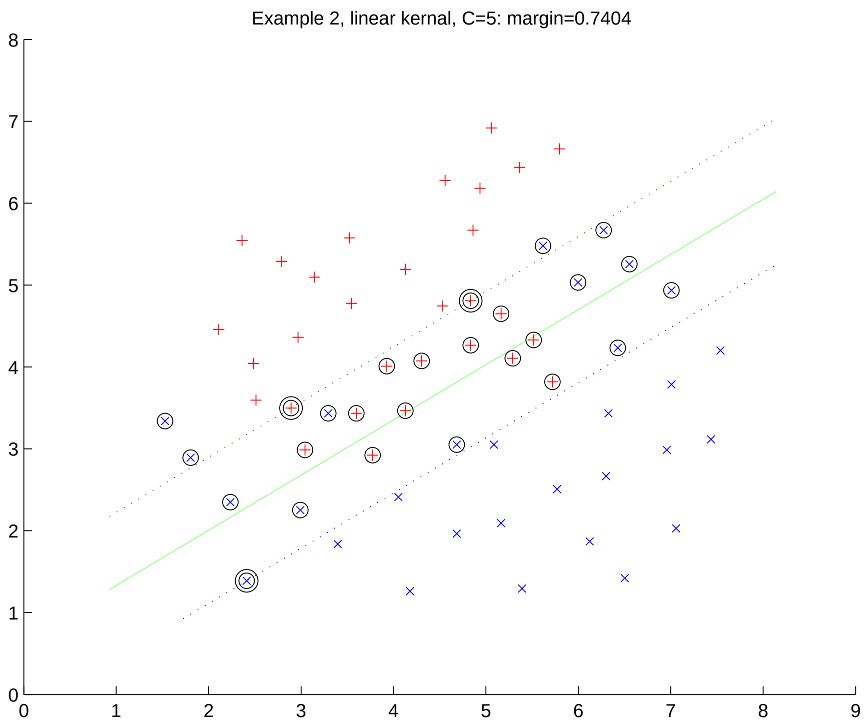
Scoring: *2 points for the plot, 2 points for the explanation*

8. Recall that we are using Lagrange multipliers to solve a minimization problem on the weights $\mathbf{w}$, where we would like to minimize $\|\mathbf{w}\|^2$ subject to $y_i(\mathbf{w}^T\mathbf{x}_i + w_0) \geq 1$ for all training examples i. In this case, no weight vector satisfies all the constraints. How is this addressed by bounding the Lagrange multipliers ?

    Answer: If some of the constraints are not satisfied, then there are i for which $y_i(\mathbf{w}^T\mathbf{x} + w_0) - 1 < 0$. The term $\alpha_i\left(y_i(\mathbf{w}^T\mathbf{x} + w_0) - 1\right)$ appearing in our objective function would thus decrease as we increase α_i. Bounding α_i in effect sets a limit as to how much having an unsatisfied constraint can influence the objective function. Bounding the Lagrange multipliers, the dual objective function becomes bounded.

    Scoring: *2 points*

9. Train a SVM using the a linear kernel, imposing a bound of $C = 5$. Turn in the `plotsvm2d` plot. How many support vectors are there ? How many training points are misclassified ? How many training points do not satisfy the SVM constraints $y_i(w^T\mathbf{x}_i + w_0) \geq 1$?



Answer:

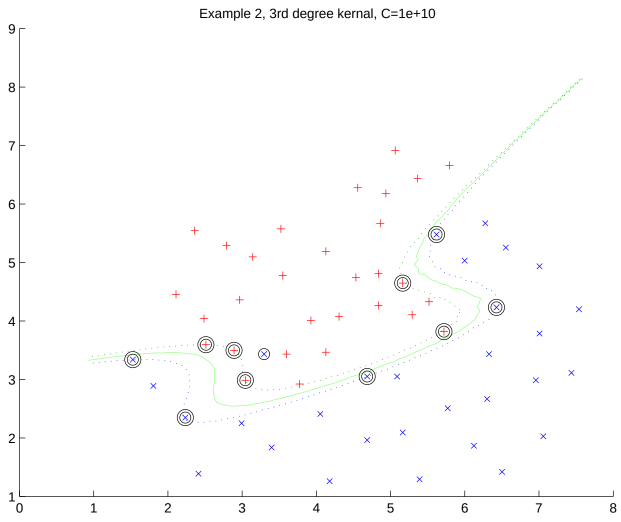
There are 26 support vectors, of which 23 do no satisfy the SVM constraints, but only 12 are misclassified.

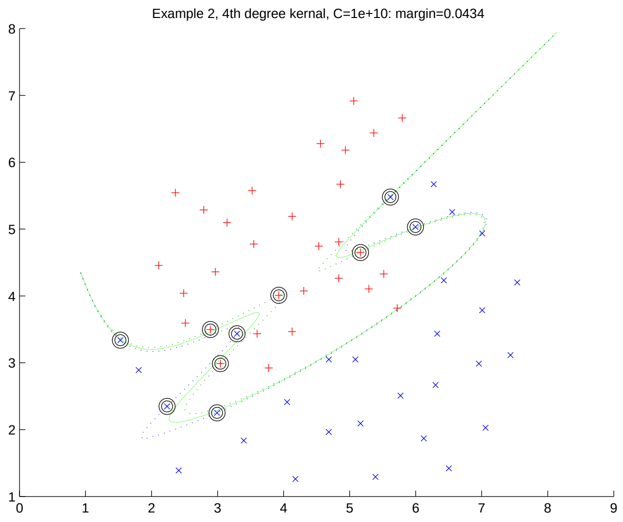
Scoring: *4 points, one for the plot, one for each number*

10. In order enrich the space of allowable classifiers, we would like to use a kernel function representing a more complex feature space. Try training a SVM, on the same example data, using polynomial kernels of the form $(1+\mathbf{x}_1^T\mathbf{x}_2)^p$, with various settings of p, and look at the resulting graphs (no need to turn them in). What is the lowest degree under which the training set is separable ?

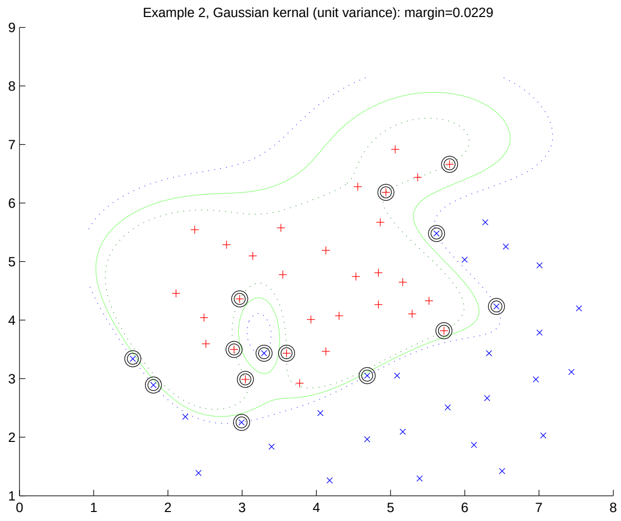
    Answer: The training data is separable using a fourth degree features.

    Scoring: *1 point*





Also try using the Gaussian kernel $e^{-\frac{|\mathbf{x}_1 - \mathbf{x}_2|^2}{2}}$ (given in `expK.m`).



11. Note that in many of the graphs you produced, the graphical margin, as it appears in the graphs, is not uniform as with the linear kernel: it is narrower in some areas and wider in others. Why does this happen ?

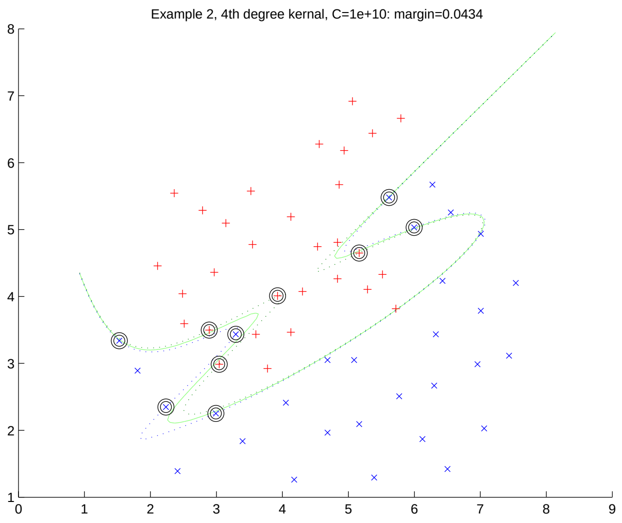
    Answer: The margin is uniform in *feature space*. What we see in *input space* is a non-linear transformation of feature space. This transformation does not conserve the width of the margin.

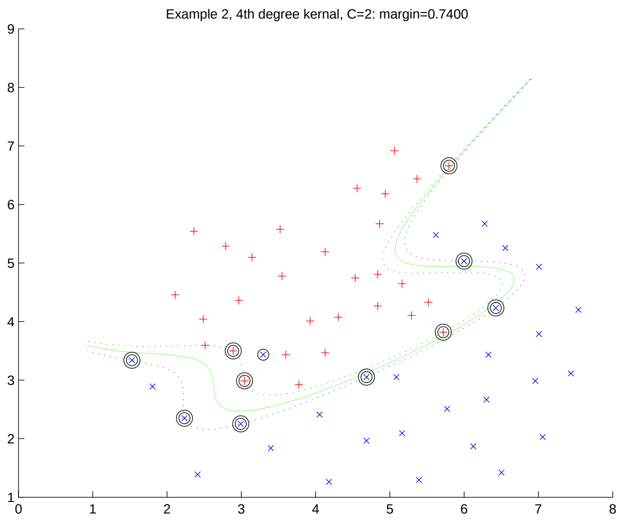
    Scoring: *2 points*

12. Even if the training set is separable, it is sometimes useful to set a bound on the values of α_i. Train a SVM using a fourth degree kernel $(1 + \mathbf{x}_1^T \mathbf{x}_2)^4$, once without a bound, and once with a bound $C = 2$. Turn in both resulting graphs, and the margin

implied by the SVM weights (i.e. the classification margins for those training points satisfying the SVM constraints).

> **Answer:** Margin without a bound is 0.0434. With a bound of $C = 2$ it increases to 0.74.





Scoring: *5 points: one for the plots, four for the margins and their calculation*

13. What is the affect of imposing a low value for C ? In this example, what happened when C was set to two ?

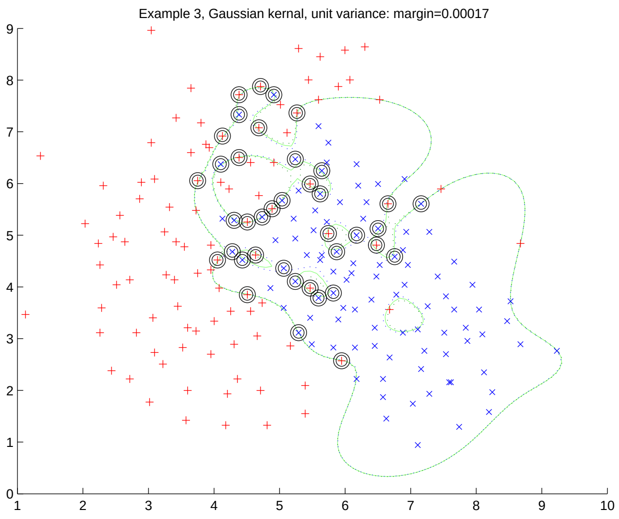
    > **Answer:** A low value of C decreases the importance of satisfying the constraints, relative to increasing the margin. We allow a few points to violate the constraints (and two points to even be misclassified), but we increase the margin more than ten-fold.

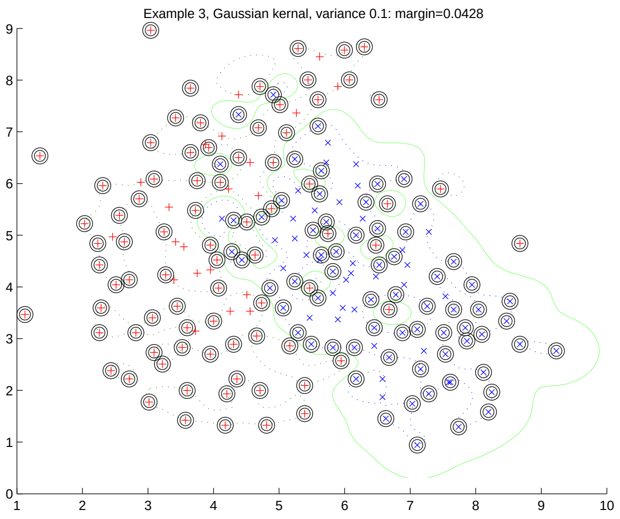
    Scoring: *2 points*

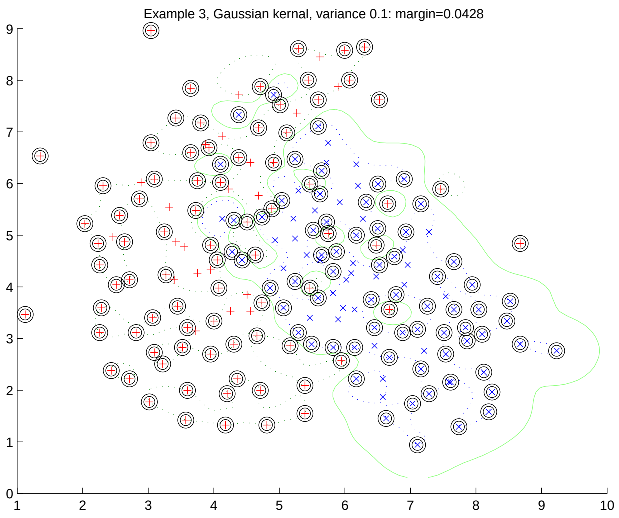
14. Using the training data `train3x`, `train3y`, investigate training a SVM with different kernels and different values of C. Is the training set separable using a Gaussian kernel ? Do you think the classification given by a SVM with a Gaussian kernel would generalize well ?

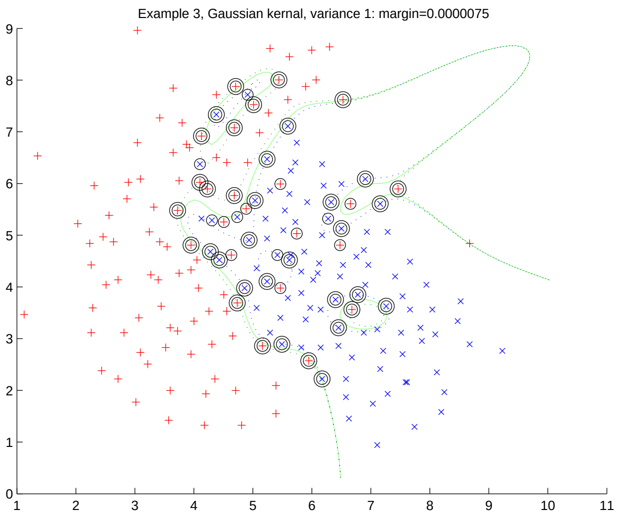
    > **Answer:** The training set is separable using a Gaussian kernel, as are all training sets (without repeated input values). Especially for small variances, almost all training points are support vectors, and so we would not necessarily expect the classification to generalize well.

    Scoring: *1 point*









Problem 3: VC-dimmeniosality

In this problem, we will investigate the VC-dimension of various sets of classifiers. In this context, a *classifier* is a function from some input space to the binary class labels +1,−1. A classifier can also be described as a subset of the input space which gets the label +1. For example, a linear classifier in the plane $\mathcal{R}^2$, can be described by a half-plane. For this reason, we can discuss the family of linear classifiers as the set of all half-planes (and possibly also the plane itself and the empty set).

We say that a class (i.e. set) $\mathcal{H}$ of classifiers *shatters* a set of points $X = \{x_1, x_2, \ldots, x_n\}$ if we can classify the points in X in all possible ways. More precisely, *for all* 2^n possible labeling vectors $y_1, y_2, \ldots, y_n \in \{-1, 1\}^n$, there exists a classifier $h \in \mathcal{H}$ such that $h(x_i) = y_i$ for all i. For any possible labeling of the points, there has to be a classifier in our set that reproduces those labels. Using the set notation for classifiers, this means that for any subset

$X' \subseteq X$ (indicating the subset of points labeled +1), there exist a classifier $h \in \mathcal{H}$ such that $X \cap h = X'$ (the set of points for which h assigns label +1 includes X' but not the rest of X). It is important to understand that shattering is a property of a set of classifiers– not of a single classifier (a single classifier cannot even shatter a single point).

The *VC-dimension* of a set $\mathcal{H}$ of classifiers is the size of the largest set X that can be shattered by $\mathcal{H}$.

We first analyze the VC-dimension of the class of axis-parallel rectangles in the plane. That is, a classifier is defined in terms of an axis-parallel rectangle, and classifies points inside the rectangle as positive, and points outside of it as negative. More formally (or perhaps just using more notation), a classifier belongs to this class if it is specified by the left, right, top and bottom coordinates and

$$h_{l,r,t,b}(x_1, x_2) = \begin{cases} +1 & \text{If } l \leq x_1 \leq r \text{ and } b \leq x_2 \leq t \\ -1 & \text{Otherwise} \end{cases}$$

Or written as a set:

$$h_{l,r,t,b} = \{(x_1, x_2) | l \leq x_1 \leq r \text{ and } b \leq x_2 \leq t\}$$

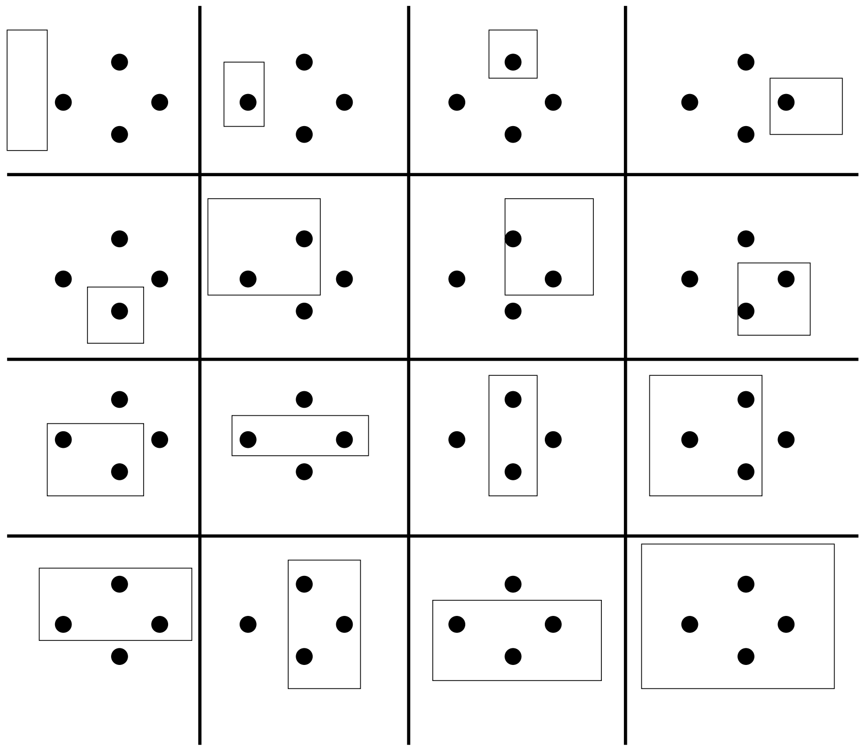
And the class of axis-parallel rectangles is

$$\mathcal{H}_{rect} = \{h_{l,r,t,b} | l < r \text{ and } b < t\},$$

which is just the set of all axis-parallel rectangles. (Your answers need not use such formal notation)

1. Find a set of four points that can be shattered by the class $\mathcal{H}_{rect}$ of axis-parallel rectangles. That is, specify four points, and show how for all 16 possible subsets (each in turn specifying the points labeled +1), you can find a rectangle that captures exactly this subset.

   Answer: Consider four points forming a diamond. The following sixteen rectangles classify the points in all possible ways:



Scoring: *3 points*

2. Prove that *no* set of five points can be shattered by $\mathcal{H}_{rect}$. The claim here is that for *any* set of five points, there exists a labeling that cannot be obtained with an axis-parallel rectangle. Conclude that the VC-dimension of $\mathcal{H}_{rect}$ is exactly four.

   Answer: For any set of five points, take a left-most point, a right-most point, a top-most point and a bottom-most point. Consider a labeling where these (at most) four points are labeled '$+$', and the remaining point(s) are labeled '-' (note that there might be more than one remaining point if a single point is, e.g. both topmost and leftmost). A rectangle containing these four extreme points must have its left edge left of the left-most point, its right edge right of the right-most point, its top edge above the top-most point and it bottom edge beneath the bottom-most point. Hence, it must contain all points, including those which we wish to label '-'. This labeling is thus unachievable with axis-parallel rectangles.

   Following this, no five points can be shattered by axis-parallel rectangle. Since we saw in the previous question how four points *can* be shattered, we can conclude that the VC dimension is exactly four.

   Scoring: *3 points*

Now consider classification of points on the one-dimensional line $\mathcal{R}^d$ using d-degree polynomials as discriminant functions. These are linear classifiers with basis functions

x^k, $k = 1, \dots, d$. Any classifier in this class has the form $h_{\mathbf{w}}(x) = \text{sign}(w_0 + w_1 x + w_2 x^2 + \ldots + w_d x^d)$. The set of all such classifiers is

$$\mathcal{H}_d = \{h_{\mathbf{w}}, \mathbf{w} \in \mathcal{R}^{d+1}\}$$

3. (*Optional*) Prove that the VC-dimension of $\mathcal{H}_d$ is $d + 1$.

   Answer: Consider the points $\{1, 2, \dots, d+1\}$. For any labeling $(y_1, y_2, \dots, y_{d+1})$, we will search for a degree d polynomial p such that $p(i) = y_i$ for all i (where y_i is $+1$ or -1). We know such a polynomial exists, since for any designated target values on $d + 1$ input values, there exists a unique degree d polynomial that satisfies all the constraints. This polynomial achieves the desired labeling. Since this is true for any labeling, the $d + 1$ points can be shattered (in fact, any $d + 1$ points can be shattered).

   We will now show that no $d+2$ points can be shattered. For any $d+2$ points, considering a desired labeling in which the points are assigned alternating labels (i.e. each to consecutive points on the real line have different labels). The polynomial must cross zero between any two points, hence it must be equal to zero in at least $d+1$ different places. But a d degree polynomial that has $d+1$ roots must be the equal to zero everywhere, and such a polynomial surely does not achieve the desired labeling (in fact, it does not achieve any labeling).

As mentioned in class, this results extends to more general types of basis functions. Consider a class $\mathcal{H}$ of classifier that can all be written as $h_{\mathbf{w}}(\mathbf{x}) = \text{sign}(w_0 + w_1\phi_1(\mathbf{x}) + w_2\phi_2(\mathbf{x}) + \cdots + w_d\phi_d(\mathbf{x}) + \phi_{\text{fixed}}(\mathbf{x}))$ for some fixed set of basis functions (by *fixed* basis functions we mean that the basis functions are the same for all classifiers, and the classifiers differ only in their weight). Note that we also allow a special term $\phi_{\text{fixed}}(\mathbf{x})$, for which we cannot vary the weight. Regardless of the basis functions, $\mathcal{H}$ has a VC-dimension of at most $d+1$. In general, if the basis functions are *independent* (i.e. no basis function is a function of other basis functions), and all weight vectors are allowed, the VC-dimension will be exactly $d + 1$.

4. Determine the VC-dimension of each of the following classes, by considering them as linear discriminant with specific basis functions. Briefly explain your reasoning in each case.

   (a) Circles in the plane. (Hint: use the basis-function formulation only as an upper bound).

   Answer: It is easy to see how to shatter three points (e.g. in an equilateral triangle) using circles.

   In order to prove that the VC dimension is not more than three, note that a point (x_1, x_2) lies inside a circle of radius r with center (a, b) iff:

$$(x_1 - a)^2 + (x_2 - b)^2 \leq r^2$$

Or equivalently:

$$x_1^2 - 2ax_1 + a^2 + x_2^2 - 2bx_2 + b^2 - r^2 \le 0$$

Collecting terms we get:

$$(-2a)x_1 + (-2b)x_2 + (a^2 + b^2 - r^2)1 + (x_1^2 + x_2^2) \le 0$$

We can now rewrite the above as a weighted combination of three features, plus a fixed-weight feature:

$$\text{sign}\left(w_1\phi_1(x_1, x_2) + w_2\phi_2(x_1, x_2) + w_3\phi_3(x_1, x_2) + \phi_{\textsf{fixed}}(x_1, x_2)\right)$$

where:

$$\begin{aligned} w_1 &= -2a & \phi_1(x_1, x_2) &= x_1 \\ w_2 &= -2b & \phi_2(x_1, x_2) &= x_2 \\ w_3 &= a^2 + b^2 - r^2 & \phi_3(x_1, x_2) &= 1 \\ & & \phi_{\textsf{fixed}}(x_1, x_2) &= x_1^2 + x_2^2 \end{aligned}$$

All circle classifiers can be written as such a weighted combination of these features, and thus the VC-dimension of circle classifiers is at most three. Note that since not all weights are possible (since r^2 must be positive), we cannot conclude from this alone that the VC-dimension is in fact three, and not lower. We can conclude this from the shattering of three points, establishing a VC-dimension of exactly three.

Scoring: *4 points*

(b) *(Optional)* Ellipsoids in the plane.

Answer: Using similar arguments, the VC-dimension of axis parallel ellipses is four, and of general ellipses it is five.

(c) Classifiers representing decision regions of 2-component Gaussian mixture models in the plane. (Hint: rewrite the class posterior as $\frac{1}{1+e^{z(\mathbf{x})}}$ and use $z(\mathbf{x})$ as a discriminant function).

Answer: We already saw how the posterior class probabilities in a Gaussian mixture model is of the form $\frac{1}{1+e^{z\mathbf{x}}}$, where $z(\mathbf{x})$ is a quadratic function. In fact, we saw how for any quadratic function $z(\cdot)$, there exists a Gaussian mixture model with a posterior of $\frac{1}{1+e^{z\mathbf{x}}}$. Based on the posterior, the class decision criterion is $\frac{1}{1+e^{z\mathbf{x}}} < 0.5$, which is equivalent to $z(\mathbf{x}) < 0$. Gaussian mixture classifiers are thus exactly the classifiers given by the sign of quadratic functions. The number of quadratic features in the plane is six: $1, x_1, x_1^2, x_2, x_2^2, x_1x_2$. Thus, the VC-dimension of Gaussian mixture models in the plane is six.

Scoring: *4 points*

(d) *(Optional)* Classifiers representing decision regions of 2-component Gaussian mixture models in R^d.

Answer: Using a similar argument, and counting the number of quadratic features in R^d: one bias term, d linear features, d squared features, $\binom{d}{2}$ quadratic features, for a combined VC-dimension of $1 + 2d + d(d-1)/2 = 1 + \frac{3}{2}d + \frac{1}{2}d^2$.

Note that this is less then the straight-forward counting of parameters in the standard parameterization of Gaussian mixture models through priors, means and the covariance matrix. Following this count we would get 2 parameters from the priors, d from each mean, and d^2 from each covariance matrix, for a combined total of $2 + 2d + 2d^2$. This is certainly an overcounting, since we know, for example, that the priors must sum up to one (hence it is enough to specify a single prior) and that the covariance matrix must be symmetric. But even after accounting for this, we are still left with $1 + 2d + d(d+11)/2 = 1 + \frac{5}{2}d + \frac{1}{2}d^2$ parameters. But we know there are still more redundancies in the parameterization, since the covariance matrix must be positive definite, and there are many different Gaussian mixture models sharing the same posterior, as well as different posteriors sharing the same decision boundary. The VC-dimension provides for a way of accounting for these further redundancies.

5. In this question we will explore the VC-dimension of decision stumps. Recall that a stump classifier $h_{i,a,b}$ in R^d is specified by an axis-parallel half-space: $h_{i,a,b} = \{(x_1, x_2, \ldots, x_n) | ax_i \leq b\}$ (a can always be taken to be $+1$ or -1).

(a) *(Optional)* Consider a $d = 2^\delta$ dimensional space. Suggest a set of δ point in $\mathcal{R}^d = \mathcal{R}^{(2^\delta)}$ that can be shattered by stumps, and show how it can be shattered. Conclude that the VC-dimension of stumps in R^d is at least $\lfloor \log_2 d \rfloor$ (i.e. $\log d$ rounded down to the nearest integer).

Answer: Each of the d coordinates will correspond to one of the $2^\delta = d$ possible labeling of our δ points. We will set value of the ith coordinate of the jth point, to $+1$ or -1 according to the labeling of the jth point under the ith labeling. To achieve any labeling, all that is needed is a stump $x_i > 0$ along the relevant coordinate i.

(b) For any set of n points in $\mathcal{R}^d$, show that the stumps can only classify the n points in at most $2dn$ different ways. That is, there are at most $2dn$ different labeling on the points which are attainable using stump classifiers. (Hint: count the number of possible classifications using stumps on a specific axis)

Answer: For each for the i coordinates, consider the ordering $\sigma = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ of points along that coordinate. Stumps along this coordinate can only separate between some prefix of σ and the corresponding suffix. There are $n+1$ prefixes (including the empty prefix and the complete set of points) and each one can either be classified as $+1$ or as -1, for a total of at most $2(n+1)$ classifications along this coordinate (not all of these might be possible, if there are repeated values in this coordinate among the n points). But note that classifying the empty prefix as $+1$ is equivalent to classifying the complete prefix as -1, and equivalently for classifying the empty prefix as -1 and the complete prefix as $+1$, reducing the number of of possible classifications to at most $2n$.

This analysis can be applied to each of the d coordinates, and so the total number of classifiers is at most $2dn$ (we are overcounting, since stumps along different coordinates may lead to the same classification).

Scoring: *2 points*

(c) Use the above bound to show that the VC-dimension of stumps is at most $2(\log_2 d + 1)$. (Hint: use the fact that $\log_2 n < n/2$ and so $n - \log_2 n > n/2$)

Answer: If the VC-dimension is n, then there must be a set of n points that is shattered. For this set of n points, all 2^n labeling must be achievable. But using the above bound we can only classify n points in at most $2dn$ different ways, hence:

$$\begin{aligned}
2^n &\le 2dn \\
n &\le \log_2(2dn) = 1 + \log_2 d + \log_2 n \\
\frac{n}{2} < n - \log_2 n &\le 1 + \log_2 d \\
n &< 2(\log_2 d + 1)
\end{aligned}$$

Scoring: *4 points*

Combining the results, we see that the VC-dimension of stumps is roughly $\log d$ (more formally, it is $\Theta(\log d)$: for large enough d, it differs from $\log d$ only by a multiplicative constant).

Although the VC-dimension of stumps is fairly low, combining stumps, as we did in AdaBoost, is much more expressive. A linear combination of m stumps is a classifier given by $h(x) = \operatorname{sign}(\alpha_1 h_1 + \alpha_2 h_2 + \cdots + \alpha_m h_m)$, where h_j are decision stumps, and $\alpha_m \in \mathcal{R}$ are their corresponding weights. We will focus on linear combinations of stumps on the one-dimensional line $\mathcal{R}$.

Note that this is not a linear combination of fixed basis functions, since the stumps h_j vary from from classifier to classifier.

(d) Show how $m/2$ points in $\mathcal{R}$ can be shattered by convex combinations of m stumps. (Hint: for each point use a combination of two stumps that classifies the point correctly, and is ambivalent on all other points). State the resulting bound on the VC-dimension of convex combinations of m stumps in the one-dimensional line.

Answer: For each point x_i which should be positively classified, consider two stumps given by $x \leq x_i + \epsilon$ and $x \geq x_i - \epsilon$, for ϵ small enough so that there is no other point within ϵ of x_i. Similarly, if x_i should be negatively classified, consider the two stumps $x \geq x_i + \epsilon$ and $x \leq x_i - \epsilon$. Let the weights of all stumps be equal. The votes of the two x_i stumps cancel out when voting on any other point x_j ($j \neq i$), and so the total classification of any point x_i is exactly the (common) vote of its two stumps on it, which is correct.

Since we can shatter a set of $m/2$ points, the VC-dimension of a convex combination of m stumps is at least $m/2$.

Note that we can also show how to shatter m points using m stumps, and thus show a lower bound of m on the VC-dimension. We can shatter m points by having a stump between each two consecutive points with non-agreeing labels, with the stump classifying these two points correctly. This will require at most $m-1$ stumps, all of which will have equal weight, and result in a combined weighted vote of either $-1/0$ or $0/1$ on points that should be classified $-1/+1$. By introducing an additional stump that classifies all points the same way, and has half the weight as the rest of the stumps, we can get correct classification on all points.

Scoring: *4 points: 3 for the example, 1 for the bound of $m/2$.*

(e) *(Optional)* For $m+2$ points in $\mathcal{R}$, show a specific labeling that cannot be attained by a convex combination of m stumps. State the resulting bound on the VC-dimension of convex combinations of m stumps in the one-dimensional line.

Answer: A labeling with alternating labels cannot be attains, since there must be at least one stump between every pair of points, for a total of at least $m+1$ stumps.

(f) *(Optional)* Bound the VC-dimension of a convex combination of m stumps in higher dimensions.

Now consider sinusoidal classifiers over the real line $\mathcal{R}$. A sinusoidal classifier h_ω is specified by:

$$h_\omega(x) = \text{sign}(\sin(\omega x))$$

And the class of sinusoidal classifiers contains all such classifiers:

$$\mathcal{H}_{\text{sin}} = \{h_\omega | \omega > 0\}$$

6. (*Optional*) Consider a set of n points $x_i = 4^{-i}$ for $i = 1, \ldots, n$. Show how this set of points can be shattered by $\mathcal{H}_{\text{sin}}$.

Answer: Suppose we are given any labeling for the n points $y_1, \ldots, y_n$. Then we set

$$\omega = \pi \sum_{i=1}^{n} \frac{(1-y_i)4^i}{2} = \pi \sum_{i:y_i=-1} 4^i$$

Suppose k is s.t. $x_k = -1$. Then we have $\omega x_k = \omega 4^{-k} = \pi(\alpha + 1 + \beta)$ Here,

$$\alpha = \sum_{i: y_i = -1 \wedge i > k} 4^{i-k}$$

$$\beta = \sum_{i: y_i = -1 \wedge i < k} 4^{i-k}$$

The 1 term comes from the cancellation of 4^{-k} with one of the 4^i terms in the sum. Note that α is even, and $\beta < 1$. Thus, $\omega x_i = \gamma + \beta$, where γ is odd. Thus, $\text{sign}(\sin(\omega x_k)) = -1 = y_k$. For k s.t. $x_k = 1$, we will get $\omega x_k = \gamma + \beta$, where γ is even, and $\text{sign}(\sin(\omega x_k)) = 1 = y_k$. Thus, $\mathcal{H}_{\text{sin}}$ can shatter the n points.

7. What is the VC-dimension of $\mathcal{H}_{\text{sin}}$? (You may use the previous question, even if you did not answered it)

   Answer: The VC-dimension is infinite, since for any n, there is a set of n points that can be shattered.

   Scoring: *1 point*

Thanks to Rui Fan and Gregory Shakhnarovich for making available their typeset solutions.