

Exercise 4.1

Let $k_q(x, z) = (x \cdot z + c)^q$, $x, z \in \mathbb{R}^n$. Consider $n = 2, q = 2, c = 1$. Now

$$k_2(x, z) = \varphi(x) \cdot \varphi(z) \quad (46)$$

$$= (x \cdot z + 1)^2 \quad (47)$$

$$= \left(\sum_i (x_i z_i) + 1 \right)^2 \quad (48)$$

$$= (x_1 z_1 + x_2 z_2 + 1)^2 \quad (49)$$

$$= x_1^2 z_1^2 + 2x_1 z_1 x_2 z_2 + 2x_1 z_1 + 2x_2 z_2 + x_2^2 z_2^2 + 1 \quad (50)$$

$$= (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1 x_2, x_2^2) \quad (51)$$

$$\cdot (1, \sqrt{2}z_1, \sqrt{2}z_2, z_1^2, \sqrt{2}z_1 z_2, z_2^2), \quad (52)$$

so $c_1 = 1, c_2 = c_3 = c_5 = \sqrt{2}, c_4 = c_6 = 1$.

rant: Doing the feature transformation explicitly by some engineered(!) $\varphi()$ has been standard statistical practice. The tradition in this regard has been to first fit a linear model (more or less our "perceptron") to the data, and if its not good enough, scrutinize the data and the model, and then try to add such nonlinearities to the representation that you have reason to believe could be fruitful, and fit another linear model to the new representation. Possibilities regarding what might be done are virtually endless. Usually e.g. power transformations, products of features, etc. are tried. Adding such new features is called *basis expansion*. The idea of constructing $\varphi()$ explicitly has the benefit that it is driven by an attempt to understand the data. In comparison, pulling some standard kernel from the sleeve and hoping for the best does not reveal much about the problem at hand. Trying to make very specific kernels for specific problems is called *kernel engineering*, which is basically just a more difficult form of basis expansion, because (atleast in theory) you'll have to make sure that the kernel is a kernel, i.e. it fulfills certain conditions. The actual underlying problem of *hypothesis class selection* does not vanish by tricks, but the representation change trick has the benefit that if you are willing to tinker with your representation or kernel, you might get quite far with just a single good algorithm to learn linear models.

Exercise 4.2

For an ANOVA kernel,

$$\varphi_A(\mathbf{x}) = \prod_{i \in A} x_i.$$

where $A \subseteq \{1, \dots, n\}$. Start by writing out the kernel,

$$k_q^n(\mathbf{x}, \mathbf{z}) = \sum_{|A|=q} \varphi_A(\mathbf{x})\varphi_A(\mathbf{z}) \quad (53)$$

$$= \sum_{|A|=q} \prod x_i z_i \quad (54)$$

$$= \sum_{|A|=q, n \in A} \prod x_i z_i + \sum_{|A|=q, n \notin A} \prod x_i z_i \quad (55)$$

$$= x_n z_n \sum_{|A|=q-1, n \notin A} \prod x_i z_i + \sum_{|A|=q, n \notin A} \prod x_i z_i \quad (56)$$

$$= x_n z_n k_{q-1}^{n-1}(x, z) + k_q^{n-1}(x, z). \quad (57)$$

The idea was to separate the sum to two parts, where the left part has all those subsets having attributes with index n .

It is apparent from this recursive formula that it can be computed by dynamic programming. The basic algorithm would have an $(q+1) \times (n+1)$ matrix k , with initialization $k(1, :) = 1, k(:, 2 : (q+1)) = 0$. Using the recursive formula, the matrix would have been filled e.g. columnwise (process all rows of a single column, then proceed to next column) and hence the algorithm would require time $O(nq)$ to process the matrix.

Exercise 4.3

This exercise again demonstrates the potential method that we've already got some familiarity with. Denote $P_t = \frac{1}{2}\|u - w_t\|^2$. All norms here are euclidean. Lets start playing with the potential difference,

$$P_t - P_{t+1} = \frac{1}{2}\|u - w_t\|^2 - \frac{1}{2}\|u - w_{t+1}\|^2 \quad (58)$$

$$= \frac{1}{2}\left(\sum_i (u_i - w_{t,i})^2 - \sum_i (u_i - w_{t+1,i})^2\right). \quad (59)$$

Now by plugging in $w_{t+1,i} = w_{t,i} - \eta(\hat{y}_t - y_t)x_{t,i}$ (note the minus sign!) and doing some tedious manipulations, we can end up with

$$P_t - P_{t+1} = (\hat{y}_t - y_t)^2\left(\eta - \frac{1}{2}\eta^2\|x_t\|^2\right). \quad (60)$$

One potential problem with the manipulations to reach the above expression is to forget to square the η when you take it out from the squared norm.

Now we'll just sum over t and observe the telescoping property, giving us

$$P_1 - P_{t+1} = \sum_t (\hat{y}_t - y_t)^2\left(\eta - \frac{1}{2}\eta^2\|x_t\|^2\right) \quad (61)$$

$$P_1 \geq \sum_t (\hat{y}_t - y_t)^2\left(\eta - \frac{1}{2}\eta^2\|x_t\|^2\right) \quad (62)$$

$$\geq \sum_t (\hat{y}_t - y_t)^2\left(\eta - \frac{1}{2}\eta^2 X^2\right) \quad (63)$$

$$\geq \sum_t (\hat{y}_t - y_t)^2\left(\eta - \frac{1}{2}\eta^2 X^2\right), \quad (64)$$

where we used the assumption that $\|x_t\| \leq X, \forall t$. By selecting $\eta = \frac{1}{X^2}$ and remembering that $P_1 = \frac{1}{2}\|u\|^2$, we get

$$\frac{1}{2}\|u\|^2 \geq \sum (\hat{y}_t - y_t)^2\left(\frac{1}{X^2} - \frac{1}{2}\frac{X^2}{X^4}\right) \quad (65)$$

$$\|u\|^2 X^2 \geq \sum (\hat{y}_t - y_t)^2. \quad (66)$$

□

Exercise 4.4 - 4.5

Here we were supposed to try out some simple algorithms and experience their behaviour on some easily understandable artificial data.

Something like the following can be learned from doing so,

- Even on a simple problem like this, learning rate can have great effect on convergence and the test results of the algorithms.
- The theorem-suggested learning rate for winnow does not appear to be particularly good for this problem. Instead, a rate of 2 seems to work well.
- Perceptron and marginalised perceptron tend to increase their test error as the dimension is increased. Winnow still works acceptably with $d = 5000$.
- The development of the $f()$ ratio suggested in the exercise gets worse and worse for perceptron and marginalised perceptron as the dimension grows. With high learning rates, $f()$ can be very erratic for Winnow.
- The lower the learning rate for Winnow is, the more it behaves like the two other algorithms. Empirically this can be seen from the plot of $f()$ or the test error behaviour.
- We do not appear to get anywhere near the mistake bounds proposed by the theorems. For example, the bounds that are inversely related to the margin can be particularly out of scale here.

For completeness, I've included a code draft. This time its in Matlab.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
  
EXC=200;  
DIM=50;  
  
algorithms={'winnow','perceptron','margperceptron'};  
  
% create data and target concept  
X=rand(EXC,DIM)*2-1;  
w=zeros(DIM,1);  
w(1:2)=0.5;  
  
% label data by concept  
labels=(X*w>=0)*2-1;  
  
% calc margin  
margin=min(abs(X*w));  
% calc norms of the data  
Xnorm = max( sqrt(sum(X.^2,2)) );  
XmaxNorm = max(max(abs(X), [], 2));
```

```

% split it to train and test
trainX=X(1:(EXC/2),:);
testX=X((EXC/2+1):EXC,:);
trainLab=labels(1:(EXC/2));
testLab=labels((EXC/2+1):EXC);

% initialize our hypotheses
wM=zeros(length(algorithms),DIM);
% winnow needs positive starting coefficients
wM(1,:)=1;

rho=margin - 0.1; % cheating a bit for marg. perc.

thresh=zeros(length(algorithms),1);
thresh(1)=0;
thresh(2)=0;
thresh(3)=rho;

alpha=zeros(length(algorithms),1);
%alpha(1)= margin/(XmaxNorm^2);
alpha(1)= 2;
alpha(2)= (Xnorm^2) / (margin^2);
alpha(3)=(margin-thresh(3))/(Xnorm^2);

% winnow bound
wbound=2*(Xnorm^2)*log(DIM) / margin^2;
fprintf(1, 'winnow bound %f\n', wbound);

% perceptron bound
pbound=(Xnorm^2)/(margin^2);
fprintf(1, 'perceptron bound %f\n', pbound);

% margperc bound
% for our reference model, hinge loss is zero due to our choice
% of mu. hence forget the first term.
mbound=1/(2*alpha(3)*(margin-thresh(3)-alpha(3)*(Xnorm^2)/2));
fprintf(1, 'marg. perceptron bound %f\n', mbound);

% learn the models
for alg=1:length(algorithms)
    doIter=1;
    f=[];
    totIters=0;totMistakes=0;
    while(doIter)
        didMistake=0;
        for i=1:size(trainX,1)
            pred=(sum(wM(alg,:).*trainX(i,:))-thresh(alg))*trainLab(i);

            if(pred<=0)
                wM(alg,:)=feval(char(algorithms(alg)),wM(alg,:), ...

```

```

                                trainX(i,:),alpha(alg),trainLab(i));
        didMistake=didMistake+1;
    end
    f=[f; (abs(wM(alg,1))+abs(wM(alg,2))) / ...
        (eps+sum(abs(wM(alg,:))))];
end

totMistakes=totMistakes+didMistake;
if(rand(1)<0.01)
    fprintf(' alg %d did %d mistakes, tot %d \n', ...
        alg, didMistake, totMistakes);
end
doIter=didMistake;
totIters=totIters+1;
end
fprintf(' alg %d reqd %d iters %d mistakes \n', ...
    alg, totIters, totMistakes);
testerr=sum( (testX*wM(alg,:)).*testLab)<0);
fprintf(' testerr %1.2f \n', testerr/length(testLab));
plot(f)
pause;
end

function RES = winnow(w,x,alpha,label)
    RES=w.*exp(label.*alpha.*x);

function RES = perceptron(w,x,alpha,label)
    RES=w+label*alpha*x;

function RES = margperceptron(w,x,alpha,label)
    RES=w+label*alpha*x;
    rnorm=sqrt(sum(RES.^2));
    if(rnorm>1)
        RES=RES./rnorm;
    end
end

```

%%%

rant: Now that we have applied Winnow on a data where there are clearly relevant and irrelevant attributes, a word of caution is in order. The *irrelevant attributes* expression, together with the property of *attribute efficiency* (such as Winnow has) are convenient theoretical abstractions adapted by the online machine learning community. There, the attributes are deemed irrelevant if they do not belong to a particular supposed target concept. This might be fine in e.g. logic (a literal either belongs to the formula or it does not), but in continuous, natural phenomena, the usefulness of some attribute might be more graded. Also, for the given concept class (like a linear classifier for the parity problem) the attributes might be irrelevant in another way, as no model of the hypothesis class is expressive enough to make use of them. In comparison, instead of speaking of relevancy as a binary phenomenon, a statistician might claim some

level of dependency to exist between feature x_i and the class label according to the model he estimated (i.e. "feature x_i can *explain* the class variable with ... and so on."). In machine learning, the various kinds of relevancy have been attempted to be conceptualized by e.g. Blum & Langley: Selection of Relevant Features and Examples in Machine Learning. *Artif. Intell.* 97(1-2): 245-271 (1997). Suffice to say, an empirist would often be in a very lucky situation if the only thing he had to do was to remove a few irrelevant variables, in order to make the learning succeed. More likely we have to do expand the basis like in exercise 4.1 and then do pruning in a potentially combinatorially exploding situation.

It should be noted that some variables might not be only irrelevant, but even downright harmful. For example, if one is learning a decision tree, and there is a complex but correct explanation, and a wrong, but very tempting and easy explanation, it might be that the only practical way to prevent the learner from going the easy way is to remove that possibility from the data (in a Bayesian setup we would make the prior probability of the easy solution a really low one or even zero). However, it might be very difficult to know when the learning method has somehow "cheated" on real world data.

Next time we will have an exercise with natural data of car images. There it is rather intuitive to imagine what properties of the data could be irrelevant/harmful with relation to the concept we really wish to learn.