

Exercise 10.1

The optimization criteria is

$$\begin{aligned} \min. \quad & -\mu + C \sum_i^m \epsilon_i^2 \\ \text{s.t.} \quad & \|\mathbf{w}\|^2 - 1 \leq 0 \\ & (-\epsilon_i \leq 0, \quad i \in 1, \dots, m) \\ & \mu - \epsilon_i - \mathbf{y}_i(\mathbf{w}_i^T \phi(x_i) - b) \leq 0, \quad i \in 1, \dots, m \end{aligned}$$

Notice that we have dropped the second constraint as it is redundant. This can be seen by noting that for all i , if $\epsilon_i < 0$, we can set $\epsilon_i = 0$ while still fulfilling the constraints and yet getting a smaller objective function.

Continue as usual by writing the lagrangian,

$$L(\mathbf{w}, b, \mu, \epsilon, \alpha, \lambda) = -\mu + C \sum_i \epsilon_i^2 - \sum_i \alpha_i (-\mu + \epsilon_i + \mathbf{y}_i(\mathbf{w}^T \phi(x_i) - b)) + \lambda (\|\mathbf{w}\|^2 - 1) \quad (176)$$

and differentiating (note that \mathbf{w} , α and ϵ are vectors) and setting to zero,

$$\frac{\delta L}{\delta \mathbf{w}} = - \sum_i \alpha_i \mathbf{y}_i \phi(x_i) + 2\lambda \mathbf{w} = 0 \quad (177)$$

$$\Rightarrow \mathbf{w} = \frac{1}{2\lambda} \sum \alpha_i \mathbf{y}_i \phi(x_i) \quad (178)$$

$$\frac{\delta L}{\delta b} = \sum a_i \mathbf{y}_i = 0 \Rightarrow \sum a_i \mathbf{y}_i = 0 \quad (179)$$

$$\frac{\delta L}{\delta \mu} = -1 + \sum a_i \Rightarrow \sum a_i = 1 \quad (180)$$

$$\frac{\delta L}{\delta \epsilon} = 2C\epsilon - \alpha \Rightarrow \epsilon_i = \frac{\alpha_i}{2C} \quad (181)$$

Inserting these back into the lagrangian and beautifying eventually yields the dual,

$$G(\alpha, \lambda) = -\frac{1}{4C} \sum \alpha_i^2 - \frac{1}{4\lambda} \sum_i \sum_j \alpha_i \alpha_j \mathbf{y}_i \mathbf{y}_j k(x_i, x_j) - \lambda, \quad (182)$$

where we have changed to the kernel notation ($k(x_i, x_j) = \phi(x_i)^T \phi(x_j)$). Denote the double summation term in the above formula as a . Now

$$\frac{\delta G}{\delta \lambda} = \frac{1}{4\lambda^2} a - 1 = 0 \Leftrightarrow \lambda = \frac{\sqrt{a}}{2}. \quad (183)$$

Note that λ must be positive as a Lagrange multiplier, and that if a were negative, the derivative wouldn't have a valid (real-valued) root. Inserting this λ back to the dual G results in the wanted solution

$$W(\alpha) = -\frac{1}{4C} \sum \alpha_i^2 - \left(\sum_i \sum_j \alpha_i \alpha_j \mathbf{y}_i \mathbf{y}_j k(x_i, x_j) \right)^{\frac{1}{2}}. \quad (184)$$

□

Exercise 10.2

a) The task is to minimize

$$G(\alpha) = \sum_i \exp(-\alpha y_i h(x_i) - y_i f(x_i)) \quad (185)$$

$$= \sum_i \exp(-\alpha y_i h(x_i)) \exp(-y_i f(x_i)) \quad (186)$$

$$= \sum_i \exp(-\alpha y_i h(x_i)) d_i \quad (187)$$

where we noticed that the exp term on the right is actually the weighting distribution d_i . Then proceed by decomposing the sum to parts where the hypothesis makes mistake or does not (since $y, h \in \{-1, 1\}$, the exp's simplify nicely),

$$= \sum_i d_i \exp(-\alpha y_i h(x_i)) \quad (188)$$

$$= \sum_{err} d_i \exp(\alpha) + \sum_{ok} d_i \exp(-\alpha) \quad (189)$$

$$= \exp(\alpha) \sum_{err} d_i + \exp(-\alpha) \left(\sum d_i - \sum_{err} d_i \right) \quad (190)$$

$$= \exp(\alpha) \sum_{err} d_i + \exp(-\alpha) \left(1 - \sum_{err} d_i \right), \quad (191)$$

which, remembering the definition of $\epsilon = \sum_{err} d_i$, is straightforwardly

$$= \exp(\alpha)\epsilon + \exp(-\alpha) - \exp(-\alpha)\epsilon. \quad (192)$$

Differentiating this gives

$$\frac{\delta G}{\delta \alpha} = \epsilon \exp(\alpha) - \exp(-\alpha) + \epsilon \exp(-\alpha). \quad (193)$$

Setting the above to zero and solving for α yields

$$\alpha = \frac{1}{2} \ln \frac{1 - \epsilon}{\epsilon} \quad (194)$$

b) Lets denote our new distribution with u . Then we have from AdaBoost that

$$e\hat{r}(h, S, u) = \sum_{i=1}^n u_i I(h(x_i) \neq y_i) = \epsilon(u) \quad (195)$$

$$u_i = \frac{1}{Z} d_i \exp(-\alpha y_i h(x_i)) \quad (196)$$

$$Z \Rightarrow \sum_i u_i = 1 \quad (197)$$

$$\alpha = \frac{1}{2} \ln \frac{1 - \epsilon(u)}{\epsilon(u)} \quad (198)$$

We need to show that $\epsilon(u) = \frac{1}{2}$. Lets insert u_i into $\epsilon(u)$ first. This results in

$$\frac{1}{Z} \sum d_i \exp(-\alpha y_i h(x_i)) I(h(x_i) \neq y_i) = \frac{\exp(\alpha)}{Z} \sum_{err} d_i. \quad (199)$$

Requiring Z such that $\sum u_i = 1$ allows us to solve for Z , decompose the error sum, and plug it in, giving

$$\frac{\exp(\alpha)}{\sum_{err} d_i \exp(\alpha) + \sum_{ok} d_i \exp(-\alpha)} \sum_{err} d_i, \quad (200)$$

which we can further manipulate to get

$$\frac{\exp(\alpha) \sum_{err} d_i}{(\exp(\alpha) - \exp(-\alpha)) \sum_{err} d_i + \exp(-\alpha)}. \quad (201)$$

Now inserting the algorithms choice for α and noting again that $\epsilon = \sum_{err} d_i$, we get a slightly formidable expression of

$$\frac{\left(\frac{1-\epsilon}{\epsilon}\right)^{\frac{1}{2}} \epsilon}{\left(\left(\frac{1-\epsilon}{\epsilon}\right)^{\frac{1}{2}} - \left(\frac{1-\epsilon}{\epsilon}\right)^{-\frac{1}{2}}\right) \epsilon + \left(\frac{1-\epsilon}{\epsilon}\right)^{-\frac{1}{2}}}. \quad (202)$$

However, by some sweating this can be simplified to $\frac{1}{2}$.

□

Exercise 10.3

Here we wish to find the closest distribution d (in the relative entropy sense) to the previous distribution q , such that the previous hypothesis has zero edge on the new distribution. This can be written as

$$\begin{aligned} \min. \quad & \sum_i d_i \ln \frac{d_i}{q_i} \\ \text{s.t.} \quad & -d_i \leq 0 \quad \forall i \\ & \sum d_i - 1 = 0 \\ & \sum d_i y_i h(x_i) = 0 \end{aligned}$$

The related lagrangian is

$$\begin{aligned} L(d, \alpha, \beta, \lambda) &= \sum d_i \ln \frac{d_i}{q_i} - \sum \alpha_i d_i + \beta (\sum d_i - 1) + \lambda \sum d_i y_i h(x_i) \\ &= \sum d_i (\ln d_i - \ln q_i - \alpha_i + \beta + \lambda y_i h(x_i)) - \beta. \end{aligned}$$

Differentiating and setting to zero gives

$$\frac{\delta L}{\delta d_i} = \ln d_i - \ln q_i - \alpha_i + \beta + \lambda y_i h(x_i) = 0 \quad (203)$$

$$\Leftrightarrow d_i = q_i \exp(\alpha_i - \beta - \lambda y_i h(x_i) - 1) \quad (204)$$

$$\Leftrightarrow d_i = q_i \exp(-\beta - 1) \exp(\alpha_i - \lambda y_i h(x_i)) \quad (205)$$

$$\Leftrightarrow d_i = q_i \exp(-\beta - 1) \exp(-\lambda y_i h(x_i)), \quad (206)$$

where we dropped α_i , since similarly to exercise 9.5, we know that $d_i > 0$ and hence $\alpha_i = 0$ for all i . From the requirement $\sum d_i = 1$ we can also use the above to solve for β , giving

$$\exp(-\beta - 1) = \frac{1}{\sum_j q_j \exp(-\lambda y_j h(x_j))} = \frac{1}{Z}. \quad (207)$$

Inserting this to the previous expression for d_i yields

$$d_i = \frac{1}{Z} q_i \exp(-\lambda y_i h(x_i)), \quad (208)$$

the AdaBoost update formula for d_i . Note that Z is a function until we fix λ . Hence, to wrap this up, we need to show that the AdaBoost's choice of λ minimizes the primal problem (maximizes the dual).

Inserting d_i , $\alpha_i = 0$ and β back to $L()$ gives the dual

$$\begin{aligned} G(\lambda) &= \sum d_i (\ln(\frac{q_i \exp(-\beta - 1) \exp(-\lambda y_i h(x_i))}{q_i}) + \beta - \lambda y_i h(x_i)) - \beta \\ &= \sum d_i (-\beta + \lambda y_i h(x_i) - 1 + \beta - \lambda y_i h(x_i)) - \beta \\ &= -\sum d_i - \beta \\ &= -1 - \beta = -\ln Z \\ &= -\ln \sum_j q_j \exp(-\lambda y_j h(x_j)). \end{aligned}$$

Now differentiating the dual w.r.t. λ yields

$$\begin{aligned}\frac{\delta G}{\delta \lambda} &= -\frac{1}{Z} \frac{\delta}{\delta \lambda} \sum_i q_i \exp(-\lambda y_i h(x_i)) \\ &= -\frac{1}{Z} \sum_i q_i (-y_i h(x_i)) \exp(-\lambda y_i h(x_i)),\end{aligned}$$

which can be zero iff $\sum_i q_i y_i h(x_i) \exp(-\lambda y_i h(x_i)) = 0$. By decomposing this sum to the cases where the hypothesis make an error and was correct, we get

$$\sum_{err} (-1) q_i \exp(\lambda) + \sum_{ok} (1) q_i \exp(-\lambda) = 0. \quad (209)$$

Then, remembering that $\epsilon = \sum_{err} q_i$,

$$-\epsilon \exp(\lambda) + (1 - \epsilon) \exp(-\lambda) = 0. \quad (210)$$

Solving this for λ yields

$$\lambda = \frac{1}{2} \ln \frac{1 - \epsilon}{\epsilon}, \quad (211)$$

as is also chosen by AdaBoost.

□

Exercise 10.4

In this and the next exercise you were supposed to experiment with an existing SVM package called `OsusVM`. With more complex machine learning algorithms (such as SVMs, decision trees, neural networks, etc...) making an efficient and practical implementation can be very time consuming. Its often a better idea to make feasibility tests (i.e. does the solution given by some type of algorithm look anything like acceptable?) with existing toolboxes if available.

I have included the code below. Its more illustrative to run it yourself and play with it, but suffice to say that a linear kernel is unsuitable for the problem, 2nd order kernel does nicely, and larger orders can overfit. Choosing the value of C is not very critical here, and proper kernel is often more important of the two (and its selection is clearly a much more difficult problem). More noise in the data would make C more important.

```
clear;

% add the SVM package to path
addpath('/home/jtlindgr/packages/osu_svm/');

load('ball.mat'); % load the provided data

for degree=[1 2 3 4];
    for C=[0.01 0.1 0.5 1 2 4 8];
        % learn a model. notice the changed data order reqd. by OsusVM
        [AlphaY,SVs,Bias,Parameters,nSV,nLabel] = PolySVC(x',y',degree,C);

        % draw a nice plot
        SVMPlot2(AlphaY,SVs,Bias,Parameters,x',y');

        % all models can be tested with the same command on osu svm
        [Label,DecisionValue] = SVMClass(x',AlphaY,SVs,Bias, ...
            Parameters,nSV,nLabel);

        % what about the achieved training error?
        err = sum(Label~=nLabel)/length(nLabel);
        fprintf(1, 'd=%d C=%f - Train err was %f\n', degree, C, err);
        pause
    end
end
```

With a linear kernel I got training errors around 40%, and the algorithm selected nearly 400 support vectors (but remember that one vector is enough to represent a linear classifier, and the solution could be converted if wished). Using a second order polynomial kernel and $C = 1$, training error was a little over 5%. Also, the required number of support vectors got smaller as C was increased. $C = 1$ resulted in only 60 support vectors used. With a very expressive kernel and large C , the training error could probably be pushed to zero - not a good idea; we'd just be fitting noise and getting worse testing errors.

□

Exercise 10.5

This exercise is similar to the previous one, with the difference that now we are considering the car dataset again, trying out RBF (Gaussian) kernel, and looking at the test error too.

The code could look as follows,

```
%%%%%%%% 10.5 %%%%%%%%%
clear;

% add the SVM package to path
addpath('/home/jtlindgr/packages/osu_svm/');

load('Lcars.mat');

% turn the car image array into an example matrix
[rows,cols,pics]=size(instances);
instances=reshape(instances,[rows*cols pics]);

% permute randomly
rp=randperm(pics);
instances=instances(rp,:);
labels=labels(rp);

% split data to train and test sets
split=floor(0.7*pics);
trainX=instances(1:split,:);
testX=instances((split+1):end,:);
trainLab=labels(1:split);
testLab=labels((split+1):end);

gammas=[0.001 0.01 0.05 0.1];
Cs=[0.5 1 2 4 8];
errmat=zeros(length(gammas),length(Cs));

didInit=0;
for i=1:length(gammas)
    for j=1:length(Cs)
        % train SVM with C=1. note the transposed inputs. for tuning
        % parameters, use "help PolySVC" etc.
        [AlphaY,SVs,Bias,Parameters,nSV,nLabel] = LinearSVC(trainX',trainLab');
        [AlphaY,SVs,Bias,Parameters,nSV,nLabel] = PolySVC(trainX',trainLab');
        [AlphaY,SVs,Bias,Parameters,nSV,nLabel] = RbfSVC(trainX',trainLab', ...
                                                         gammas(i),Cs(j));

        % training error
        [Label,DecisionValue] = SVMClass(trainX',AlphaY,SVs,Bias, ...
                                         Parameters,nSV,nLabel);
        trainErr = sum(Label~=trainLab')/length(trainLab);
```

```

% prediction error on test set
[Label,DecisionValue] = SVMClass(testX',AlphaY,SVs,Bias, ...
                                Parameters,nSV,nLabel);
testErr = sum(Label~=testLab')/length(testLab);

fprintf(1, 'For gamma=%f, C=%f, trainerr %f, testerr %f...\n', ...
        gammas(i), Cs(j), trainErr, testErr);
errmat(i,j)=testErr;
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

In this problem, the choice of the kernel parameter γ is critical. Small values as in the code above usually work well, whereas with larger values the training error is easy to get to zero, but the testing accuracy starts to resemble random guessing. Using $\gamma = 0.001$ and $C = 1$, test error around 3% can be reached, with fluctuation due to the random permutations of the data. Note that although the achieved error is generally better than what we previously got with linear perceptrons (that was around 6%), the critical notes about the dataset made in reference solutions of exercise 5.4-5.5 still apply.

rant: In this mostly theoretical course we have not too seriously considered empirically comparing the solutions returned by the learning algorithms. In some situations the comparisons should be done more properly using established statistical methods, i.e. calculate statistical significance of differences in error rates (if any), analyze error variances, consider importance of the type and amount of errors to the application domain, make very sure that the test set is not contaminated by parameter tuning or the training set data, and so on. Such practices, however, could merit a course of their own.

□