

### Exercise 5.1

a) Write out the error,

$$\begin{aligned} \text{err}(h; p) &= P(h(x) \neq f_*(x))P(f(x) = y) \\ &\quad + P(h(x) = f_*(x))P(f(x) \neq y) \\ &= P(h(x) \neq f_*(x))(1 - \eta) \\ &\quad + P(h(x) = f_*(x)) * \eta \\ &= P(h(x) \neq f_*(x))(1 - 2\eta) + \eta, \end{aligned}$$

from which we can see that the only possibility to reduce the error is to reduce the probability of disagreeing with  $f_*$ . Clearly  $f_*$  is the best choice.

b) Let  $f_*$  be the classifier that predicts the most probable class for each  $x$  according to  $P$ . Suppose that  $h \neq f_*$ , i.e.  $h$  sometimes disagrees on some instances. This means that  $h$  occasionally predicts the less likely class. It is clear  $\text{err}(h; P) > \text{err}(f_*; P)$ .

This can also be done formally e.g. by writing out the expected zero-one loss and minimizing it. This leads to selection of the label with smaller probability of error w.r.t.  $P$ .

c) This time, minimize the square loss,

$$\begin{aligned} E(L_2) &= E[(f(x) - y)^2] \\ &= \int_x \int_y (f(x) - y)^2 P(x, y) dx dy \\ &= \int_x \int_y (f(x) - y)^2 P(y|x) P(x) dx dy \\ &= \int_x P(x) \int_y (f(x) - y)^2 P(y|x) dy dx \\ &= \int_x P(x) E_{y|x} [(f(x) - y)^2] \\ &= E_x E_{y|x} [(f(x) - y)^2], \end{aligned}$$

where we transformed expectations to integrals and then used the basic knowledge from analysis to play with them. The result can be minimised pointwise, i.e. select

$$f(x) = \text{argmin}_c E_{y|x} [(c - y)^2 | X = x],$$

which equals selecting  $h(x) = f(x) = E[y|X = x]$ . Writing the expectation out in this particular case gives  $h(x) = (1)P(y = 1|x) + (-1)P(y = -1|x)$ .  $\square$

**Exercise 5.2**

Basically by rewriting theorem 3.4 of the lecture notes.

Let  $p = \text{err}(h)$ . Select  $\tilde{k} = \min\{k \mid 1 - \text{Bin}(k, m, \text{err}) \leq \delta\}$ . Note that  $\forall k$ ,

$$P(\text{me}\hat{r}(h; S) \geq \tilde{k}) = 1 - \text{Bin}(\tilde{k}, m, p) = \delta,$$

from which follows  $P(\text{me}\hat{r}(h; S) \leq \tilde{k}) \geq 1 - \delta$ . Now from definition of  $\tilde{k}$ ,

$$\begin{aligned} \text{me}\hat{r}(h; S) \leq \tilde{k} &\Leftrightarrow 1 - \text{Bin}(\text{me}\hat{r}(h; S), m, p) \geq \delta \\ &\Leftrightarrow \text{Bin}(\text{me}\hat{r}(h; S), m, p) \leq 1 - \delta \\ &\Leftrightarrow \overline{\text{Bin}}(\text{me}\hat{r}(h; S), m, 1 - \delta) \leq p. \end{aligned}$$

Hence

$$P(\overline{\text{Bin}}(\text{me}\hat{r}(h; S), m, 1 - \delta) \leq p) \geq 1 - \delta$$

and finally

$$P(\overline{\text{Bin}}(\text{me}\hat{r}(h; S), m, 1 - \delta) \geq p) \leq \delta.$$

Changing  $\overline{\text{Bin}}$  to  $\text{Bin}$  form and using Hoeffding bound for it should give us

$$\hat{\epsilon} \leq \hat{e}\hat{r}(h; S) - \sqrt{\frac{1}{2m} \ln \frac{1}{\delta}}.$$

□

**Exercise 5.3**

From the lecture notes and the previous exercise we know

$$P_1 = P(e(h) \geq \hat{e}(h; S) + \sqrt{\frac{1}{2m} \ln \frac{1}{\delta}}) \leq \delta$$

$$P_2 = P(e(h) \leq \hat{e}(h; S) - \sqrt{\frac{1}{2m} \ln \frac{1}{\delta}}) \leq \delta$$

Choose  $\delta' = \frac{\delta}{2|H|}$ . The probability of violating the first bound by some hypothesis is

$$\bigcup_{h \in H} P_1 \leq \sum P_1 \leq \sum \delta' = \sum_{|H|} \frac{\delta}{2|H|} = \frac{\delta}{2}.$$

Doing the same approximation for the lower bound we get that the probability of some hypothesis violating either bound is at most  $\frac{\delta}{2} + \frac{\delta}{2} = \delta$ . Hence, the probability that no hypothesis violates the bounds is  $\geq 1 - \delta$ . Now we have new bounds similar to those in  $P_1$  and  $P_2$  but for all the hypothesis. These two can now be combined as

$$P(|err(h; P) - \hat{err}(h; S)| \leq \sqrt{\frac{1}{2m} \ln \frac{2|H|}{\delta}}) \geq 1 - \delta.$$

□

### Exercise 5.4-5.5

In this exercise we can see that the simple perceptron algorithm does relatively well on a set of high-dimensional "natural data" if we are just looking at measures like the test error.

Running the algorithm as suggested, you should get test errors around 6% and convergence usually before 20 iterations. The early stopping criteria will usually pick a hypothesis from some iteration in the first half of the iterations. If the norms of the perceptrons are examined, these can be seen to grow each iteration, which can sometimes be interpreted as overfitting (i.e. in a vague sense the algorithm is just memorizing the training examples and losing the ability to generalize). However, the early stopping method does not appear to reduce the test error in any significant manner in this case. The method can be seen as a heuristic that sometimes works (for example when training feedforward neural networks with backpropagation), but your mileage might vary.

If visualized, the final hypothesis should resemble a car, you can see a form of the chassis and the tires. It could be interpreted as a prototype.

We were also asked to compute the test set bound. The test set bound could be easier to understand if you visualize the `Bin()` density as a function of  $p$  with a fixed  $m$  (number of trials == test set size). Now we are interested in finding the largest  $p$  (true error rate, the mean of the distribution) that fulfills our requirement for left tail area  $\delta$  (right tail area is  $1 - \delta$ ). Increasing  $p$  equals moving the mode of the reference distribution until we find a location of a good fit to what we experienced empirically.

Test set bound can be plotted against  $\delta$ , for example. Or you can give just some value, such as 0.05, and get rather reasonable claims from the bound, such as  $P(\text{true\_err} \geq 0.09) < 0.05$ .

The code for R follows.

```
#####  
  
library('pixmap'); # for plotting a greyscale image  
load('Lcars.RData'); # contains instances, labels  
  
# lets specify the test set bound function first  
testSetBound<-function(testErr,testSetSizeM,delta,step=0.01) {  
# note that Bin() is just the standard 'cumulative distribution function'  
# of the binomial distribution. This is already in R as pbinom(). Then  
# we do just a lazy mans search with a specified granularity "step".  
# note how R allows specifying default values in the function definition  
# (in contrast to matlab).  
  
failedExamples<-floor(testErr*testSetSizeM);  
  
p<-0;go<-TRUE;tmp<-delta;  
while(tmp>=delta) {  
tmp<-pbinom(failedExamples,testSetSizeM, p);  
p<-p+step;  
}
```

```

    }
    return(p-step);
}

# ok, the actual learning script

dims<-dim(instances); # (rows,cols,instance#)
X<-matrix(data=0,nrow=length(labels),ncol=prod(dims[1:2]));
# turn it to a matrix
for(i in 1:length(labels)) {
  X[i,]<-as.vector(instances[,i]);
# X[i,]<-X[i,]-mean(X[i,]); # standardization is standard practice
# X[i,]<-X[i,]/sd(X[i,]); # for certain methods, see what happens here...
}

# sample train,val,test indexes
idxsleft<-1:length(labels);
trainidxs<-sample(idxsleft,600,replace=FALSE);
idxsleft<-setdiff(idxsleft,trainidxs);
valididxs<-sample(idxsleft,199,replace=FALSE);
testidxs<-setdiff(idxsleft,valididxs);

# split the data and labels to train and test sets
trainX<-X[trainidxs,];
valX<-X[valididxs,];
testX<-X[testidxs,];
trainLab<-labels[trainidxs];
valLab<-labels[valididxs];
testLab<-labels[testidxs];

wm<-rep(0,dim(X)[2]); # initial hypothesis
wstack<-NULL; # our stored hypotheses
eta<-1; # learning rate coeff.

# loop the perceptron training until no mistakes are made
doIter<-1;totalMistakes<-0;
while(doIter) {
  mistakesNow<-0;
  cat('Starting iter ', doIter, '... ');
  for(i in 1:NROW(trainX)) {
    pred<-as.numeric(sum(wm*trainX[i,])>=0)*2-1;
    if(pred!=trainLab[i]) { # update on mistake
      wm<-wm + trainLab[i]*eta*trainX[i,] ;
      mistakesNow<-mistakesNow+1;

      # visualize the current model
      plot(pixmapGrey(matrix(data=wm,nrow=dims[1],ncol=dims[2])));
      Sys.sleep(0.01);
    }
  }
}

```

```

wstack<-rbind(wstack,matrix(data=wm,nrow=1)); # append current hypothesis

totalMistakes<-totalMistakes+mistakesNow;
normNow<-sqrt(sum(wm^2));
cat('norm', normNow, ', made', mistakesNow, ' mistakes now, ',
    totalMistakes, 'total.\n');
doIter<-(mistakesNow>0)*(doIter+1);
}

# pick the best one according to validation set
minErr<-1;pickIdx<-1;
for(i in 1:dim(wstack)[1]) {
  tmpErr<-sum(as.numeric(valX%%wstack[i,]>=0)*2-1 != valLab)
    / length(valLab) ;
  if(tmpErr<minErr) {
    pickIdx<-i;
    minErr<-tmpErr;
  }
}
cat('picked hypothesis # ', pickIdx, 'of', dim(wstack)[1], '\n');
wm<-wstack[pickIdx,];

# show the final model
plot(pixmapGrey(matrix(data=wm,nrow=dims[1],ncol=dims[2])));

# calculate final training and test set errors
trainErr<-sum(as.numeric(trainX%%wm>=0)*2-1 != trainLab)
  / length(trainLab) ;
testErr<-sum(as.numeric(testX%%wm>=0)*2-1 != testLab)
  / length(testLab) ;

cat('train: ', trainErr, 'test: ', testErr,'\n');

# what does the test set bound say?
delta<-0.05;
tSB<-testSetBound(testErr, length(testLab), delta);
cat('TSB claims P( true_error >= ', tSB, ') < ',delta,'\n');

#####

```

**rant:** Sometimes machine learning has been motivated by potentially allowing the machines to learn how to do things that we don't exactly know how to code in. Recognition of patterns (like cars, or characters) is such a problem, and sometimes machine learning methods have been proposed for such problems, or, atleast the methods have been marketed on the basis of their ability to handle such problems to some extent.

One educative aspect of this exercise is the possibility to look at such data and

the related classification problem. We can visualize the data, have an intuitive interpretation for it, and we can visualize the hypothesis. Then we can ask, if the method really is even close solving the problem or not? We should be able to see that this can not be the case.

The first reason is that the dataset is not a representative sample from any realistic application situation. In the case where we wish to detect a car, we are either having photographs from various sources, or photographs from a camera that can either be in a fixed location or mounted on a moving platform (such as a robot). In the realistic case, we get for example the following problems

- The classes are not nearly balanced  
(Note that a 6% error can be disastrous in e.g. 25 frames/sec operation or a sliding window approach.)
- The cars are not necessarily centered
- The cars are not necessarily sideviews
- The cars are not necessarily vertical
- The cars can be monster trucks
- The cars can be lighted oddly
- The cars can be partly occluded by other objects
- ...

The dataset used in this exercise doesn't reflect these problems. Instead of being *adversarial*, it is almost *beneficial*. Even if it did adequately represent reality, a linear algorithm on raw data could not solve this problem (for example, inverting the pixel values of a car image will not make it a non-car, but will make the prediction fail).

Often machine learning algorithms are run on data that we do not understand. There is not necessarily a-priori reason to believe that all kinds of similar problems (and many others) were absent from such data. Analysing the data and the domain, perhaps using machine learning methods as a guiding tool, may be required if the problem is to be solved. Especially when attempting to do industrial applications, it should be made very certain that the supplied data reflects the problem adequately.

As a final note, the dataset also shows how on some problems there are no low-level irrelevant attributes, but that the relevance can depend on the values of surrounding attributes. Visual datasets can often also allow the classifier to be statistically misled, i.e. it can learn to classify the surroundings instead of the objects. For example, learning to recognize humans in office environment and sheep on pasture will lead your average method to say that a human on a green pasture is definitely a sheep.