Data Mining with R

Linear Classifiers and the Perceptron Algorithm

Hugh Murrell

references

These slides are based on a notes by **Cosma Shalizi** and an essay by **Charles Elkan**, but are self contained and access to these documents is not essential.

for further background **wikipedia** is a useful source of information.

assumptions

To introduce classifiers we will assume that $\vec{x}$ is a vector of real-valued numerical input **features**. There are p of them.

The variable we are trying to learn is called the **response**, Y.

Inititially we will assume that Y is a **binary** class and it will simplify the book-keeping to say that the two classes are $+1$ and -1.

In linear classification, we seek to divide the two classes by a linear separator in the feature space.

If $p = 2$ the separator is a line, if $p = 3$ it is a plane, and in general it is a $p - 1$ dimensional hyperplane in a p dimensional space;

some geometry

We can specify the orientation of a plane by a vector $\vec{w}$ in the direction perpendicular to it.

There are infinitely many parallel planes with the same orientation.

We need to pick out one of them, which we can do by fixing its distance from the origin, b say.

R code to generate data separated by:

$$X2 = X1 + \frac{1}{2}$$

```
> x1 <- runif(30,-1,1)
> x2 <- runif(30,-1,1)
> x <- cbind(x1,x2)
> Y <- ifelse(x2>0.5+x1,+1,-1)
> plot(x,pch=ifelse(Y>0,"+","-"),
+      xlim=c(-1,1),ylim=c(-1,1),cex=2)
> abline(0.5,1)
> points(c(0,0),c(0,0),pch=19)
> lines(c(0,-0.25),c(0,0.25),lty=2)
> arrows(-0.3,0.2,-0.4,0.3)
> text(-0.45,0.35,"w",cex=2)
> text(-0.0,0.15,"b",cex=2)
```

data with linear separator

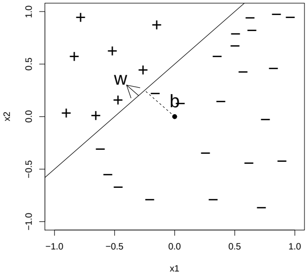

Figure: Example of a linear separator: (+ and -) indicate points in the positive and negative classes. The perpendicular vector $\vec{w}$ fixes the orientation of the separator. The directed distance b from the origin fixes the position of the separator.

using a separator to predict class

Given any separator, $(\vec{w}, b)$, we would like a formula to express the idea that points on one side of the separator are predicted to belong to one class whilst those on the other side are predicted to belong to a different class.

We can do this via:

$$\hat{Y}(\vec{x}, b, \vec{w}) = \mathrm{sign}(b + \vec{x} \cdot \vec{w})$$

where, as a reminder, $\vec{x} \cdot \vec{w} = \sum_{i=1}^{p} x_i w_i$, is the dot product and sign is a function which returns $+1$ if its argument is > 0 and -1 if its argument is < 0.

Remember that if $\vec{w}$ is a unit vector, then $\vec{x} \cdot \vec{w}$ is the projection of $\vec{x}$ on to $\vec{w}$, so this is in fact checking whether $\vec{x}$ is on the positive side of the separating plane or not.

R code for predicting class

```
> distance.from.plane = function(z,w,b) {
+   sum(z*w) + b
+ }
> classify.linear = function(x,w,b) {
+   distances =
+     apply(x, 1, distance.from.plane, w, b)
+   return(ifelse(distances < 0, -1, +1))
+ }
```

trying it out on our sample data

Our sample data was randomly selected on either side of the straight line $x2 = x1 + \frac{1}{2}$.
So in this case $\vec{w} = (-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$ and $b = -\frac{\sqrt{2}}{4}$ is a separator.

Note that b is negative because the origin is on the opposite side of the separator with respect to $\vec{w}$.

```
> classify.linear(x,c(-1,1)/sqrt(2),-sqrt(2)/4)

 [1] -1  1 -1  1  1 -1 -1 -1  1 -1 -1 -1
[13] -1 -1 -1 -1  1 -1 -1 -1  1 -1 -1 -1
[25] -1  1 -1  1 -1 -1

> Y

 [1] -1  1 -1  1  1 -1 -1 -1  1 -1 -1 -1
[13] -1 -1 -1 -1  1 -1 -1 -1  1 -1 -1 -1
[25] -1  1 -1  1 -1 -1
```

the peceptron algorithm

In 1958 Frank Rosenblatt created much enthusiasm when he published a method called the *perceptron algorithm* that is guarateed to find a separator in a separable data set.

The basic algorithm is very simple, assuming the separator passes through the origin and that the training labels Y_i are either -1 or $+1$:

```
initialize $\vec{w} = 0$
while any training observation $(\vec{x}, Y)$ is not classified correcty
    set $\vec{w} = \vec{w} + Y\vec{x}$
```

Incorrectly classified training vectors are simply added to (or subracted from) the separator normal, $\vec{w}$.

the peceptron algorithm

If the *passing through the origin* assumption is dropped then a correction to the b parameter must also be included for the peceptron algorithm to deliver a valid separator.

for incorrectly classified training samples b is adjusted according to:

set $b = b + YR^2$

where R is some constant larger than the distance from the origin to the furtherest training sample.

full peceptron code

```
> euclidean.norm = function(x) {sqrt(sum(x * x))}
> perceptron = function(x, y, learning.rate=1) {
+   w = vector(length = ncol(x)) # initialize w
+   b = 0 # Initialize b
+   k = 0 # count updates
+   R = max(apply(x, 1, euclidean.norm))
+   made.mistake = TRUE # to enter the while loop
+   while (made.mistake) {
+     made.mistake=FALSE # hopefully
+     yc <- classify.linear(x,w,b)
+     for (i in 1:nrow(x)) {
+       if (y[i] != yc[i]) {
+         w <- w + learning.rate * y[i]*x[i,]
+         b <- b + learning.rate * y[i]*R^2
+         k <- k+1
+         made.mistake=TRUE
+       }
+     } }
+   s = euclidean.norm(w)
+   return(list(w=w/s,b=b/s,updates=k))
+ }
```

Note that because a separator $(\vec{w}, b)$ classifies identically to any *scaled* counterpart $(s\vec{w}, sb)$, we choose to scale the separator so that the normal, $\vec{w}$, is returned as a **unit vector**.

testing the peceptron code

```
> (p <- perceptron(x,Y))
$w
       x1         x2
-0.8649020  0.5019408
$b
[1] -0.3760319

$updates
[1] 48
> y <- classify.linear(x,p$w,p$b)
> sum(abs(Y-y))
[1] 0
```

the separator returned by the peceptron

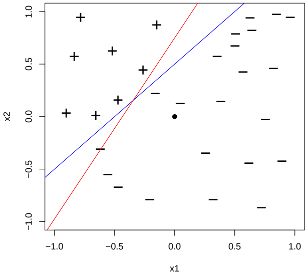

Figure: our separable data set with the original separator, $X2 = X1 + \frac{1}{2}$, shown in blue whilst the separator returned by the peceptron is shown in red.

convergence of the peceptron algorithm

The perceptron algorithm is mathematically important because of a theorem about its convergence, due to Novikoff in 1962.

Assumptions: Let $R = \max ||\vec{x_t}||$ and suppose that the learning task is solvable via a separator that passes through the origin. i.e. there exists some vector $\vec{w^*}$ of unit length and some $\delta > 0$ such that $Y_t(\vec{w^*} \cdot \vec{x_t}) > \delta$ for all t.

Theorem: Under these assumptions, the perceptron algorithm converges after at most $(\frac{R}{\delta})^2$ updates.

proof ...

Let $\vec{w}_n$ be the $\vec{w}$ vector after n updates and let $\vec{w}_0 = 0$. We will argue that whenever $\vec{w}$ is updated it becomes closer to $\vec{w}^*$. Suppose $\vec{w}_{n+1}$ is an update, i.e. $\vec{w}_n$ fails to classify an $\vec{x}$ correctly and hence $\vec{w}_{n+1} = \vec{w}_n + y\vec{x}$. Consider:

$$\begin{aligned}
\vec{w}_{n+1} \cdot \vec{w}^* &= (\vec{w}_n + y\vec{x}) \cdot \vec{w}^* \\
&= \vec{w}_n \cdot \vec{w}^* + y\vec{x} \cdot \vec{w}^* \\
&\geq \vec{w}_n \cdot \vec{w}^* + \delta
\end{aligned}$$

This tells us that the projection of $\vec{w}_{n+1}$ onto $\vec{w}^*$ has increased. We would like this to mean that $\vec{w}_{n+1}$ is closer to $\vec{w}^*$. However, what it really means is that **either** $\vec{w}_{n+1}$ is closer to $\vec{w}^*$ **or** $\vec{w}_{n+1}$ has simply grown larger.

proof continued ...

Consider the Euclidean length of $\vec{w}_{n+1}$:

$$\begin{aligned}
||\vec{w}_{n+1}||^2 &= ||\vec{w}_n + y\vec{x}||^2 \\
&= ||\vec{w}_n||^2 + 2y(\vec{w}_n \cdot \vec{x}) + ||\vec{x}||^2 \\
&\leq ||\vec{w}_n||^2 + R^2 \qquad (\text{since} \quad y(\vec{w}_n \cdot \vec{x}) \leq 0)
\end{aligned}$$

Thus, after N actual updates we know two facts: $||\vec{w}_N||^2 \leq NR^2$ and $\vec{w}_N \cdot \vec{w}^* \geq N\delta$. Putting these together:

$$N\delta \leq \vec{w}_N \cdot \vec{w}^* \leq ||\vec{w}_N|| \leq R\sqrt{N} \quad \text{and so} \quad \sqrt{N} \leq \frac{R}{\delta}$$

which means N is bounded and updates must cease eventually.

example data set

When the target variable has more than two classes the perceptron may still be used to generate a classification rule by repeatedly separating on a **one versus the rest** basis.
To demonstrate we will make use of the classic **iris** data set from R's datasets collection.

```
> data(iris)
> dim(iris)    # 150 measurements of 5 attributes
```

```
[1] 150   5
```

```
> names(iris)
```

```
[1] "Sepal.Length" "Sepal.Width"
[3] "Petal.Length" "Petal.Width"
[5] "Species"
```

target attribute: Species

The `Species` attribute in the iris dataset is categorical.

```
> summary(iris$Species)
```

```
    setosa versicolor  virginica
        50         50         50
```

The remaining four iris attributes are real valued descriptors. Can we use these real valued attributes to predict iris species? First we consider all possible bi-variate scatter plots.

iris scatter plots

```
> data(iris)
> pairs(iris[,1:4], col=iris$Species)
```

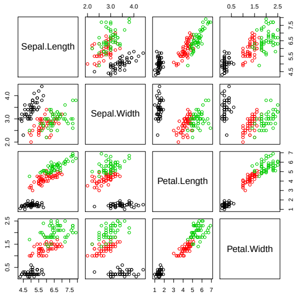

Figure: all possible bivariate scatter plots for the `iris` dataset.

find one bivariate plot with one separable species

```
> # select the Sepal.Width versus Petal.Width scatter
> x <- cbind(iris$Sepal.Width,iris$Petal.Width)
> # label setosa as positive and the rest as negative
> Y <- ifelse(iris$Species == "setosa", +1, -1)
> #
> # plot all the points
> plot(x,cex=0.2)
> # use plus sign for setosa points
> points(subset(x,Y==1),col="black",pch="+",cex=2)
> # use minus sign for the rest
> points(subset(x,Y==-1),col="red",pch="-",cex=2)
```

setosa is separable

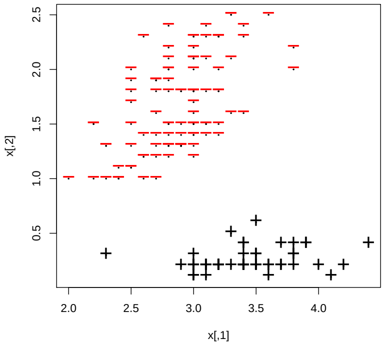

Figure: Scatter of Sepal.Width versus Petal.Width for setosa versus all other species

train our perceptron

```
> ( p <- perceptron(x,Y) )

$w
[1]  0.3277371 -0.9447690
$b
[1] -0.2543709

$updates
[1] 202

> sum(abs(classify.linear(x,p$w,p$b) - Y))

[1] 0
```

replot and view the separation boundary

```
> plot(x,cex=0.2)
> points(subset(x,Y==1),col="black",pch="+",cex=2)
> points(subset(x,Y==-1),col="red",pch="-",cex=2)
> # compute intercept on y axis of separator
> # from w and b
> intercept <- - p$b / p$w[[2]]
> # compute slope of separator from w
> slope <- - p$w[[1]] /p$ w[[2]]
> # draw separating boundary
> abline(intercept,slope,col="green")
```

separation boundary

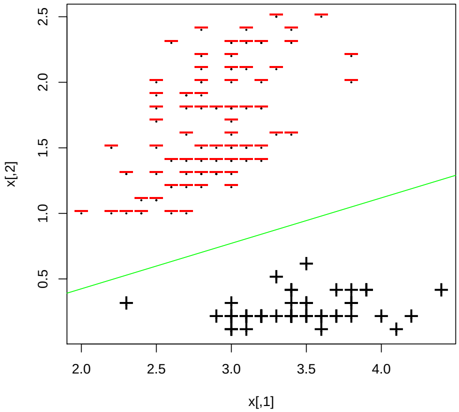

Figure: Peceptron separation of `setosa` from all other species

non-separable data

We now have one rule via `classify.linear(x,w,b)` to check if an observation is of species `setosa` or not.

If it is not then we need one more rule to separate `versicolor` from `virginica`.

A study of the all pairs `iris` scatterplots should convince you that no pair of attributes will deliver a linear separation boundary for `versicolor` from `virginica`.

It would be nice if the peceptron algorithm produced the **best** possible linear separating line in cases of non-separable observations.

Unfortunately this does not happen because the peceptron enters an endless cycle of updates when perfect separation is not possible.

exercises

- **code:** Create an R script for the peceptron code presented in these slides. Your script should include the code for the peceptron as well as testing code.
- **prove:** that the decision boundary generated by $(\vec{w}, b)$ is identical to the decision boundary generated by $(s\vec{w}, sb)$ for any scaling parameter s.
- **plot:** add a `peceptron.plot(x,y,w,b)` function to your R script that in the two dimensional case, generates a scatter plot of the training data with a superimposed peceptron decision boundary. Include code to test your plot function.
- **non-separable:** add parameters and code to your peceptron algorithm that causes the peceptron to terminate after a user specified number of updates and then returns the best decision boundary discovered so far.

exercises ...

- **iris classification:** use your enhanced peceptron to complete the iris classification problem outlined in these slides by generating a second decision boundary similar to the one shown on the next slide.

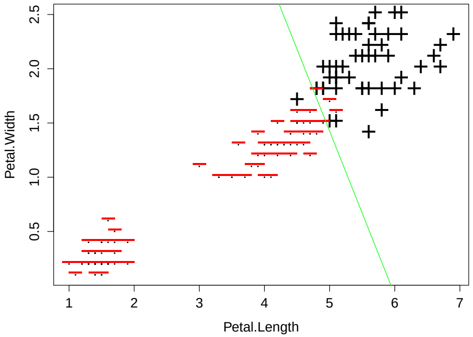

Figure: Peceptron separation of `virginica` from all other species