Polynomials

Contents

From Azzalini and Scarpa, Chapter 3

A simple prototype problem

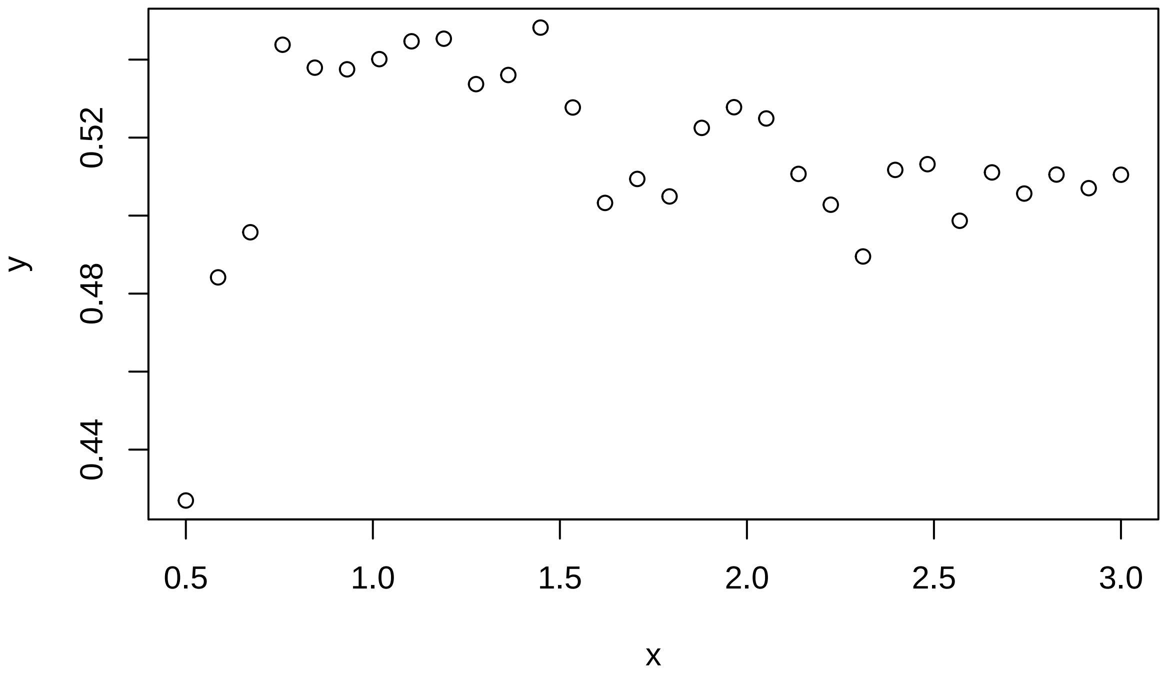
We consider here a very simple example serving as the prototype for much more complex and realistic circumstances. Let us presume that yesterday we observed $n = 30$ pairs of data, the **training set**

$$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$$

```
# import data from the web
library(readr)
df <- read_table2("http://azzalini.stat.unipd.it/Book-DM/yesterday.dat")[-31,]
# create training set
train <- data.frame(x=df$x, y=df$y.yesterday)
```

The data are shown in the scatterplot:

```
plot( y ~ x , train)
```



The data were generated artificially by an equation such as

$$y = f(x) + \varepsilon$$

where ε is an error component with distribution $N(0, \sigma^2)$ and $\sigma = 10^{-2}$; $f(x)$ is a function which we leave unspecified - the only requirement is that this function should follow an essentially regular trend.

Clearly, to generate the data, we had to choose a specific function (not a polynomial), but we do not disclose our choice.

Say we wish to obtain an estimate of $f(x)$ today that allows us to predict y as new observations of x become available, i.e. we need to fit a predictive model

$$\hat{y} = \hat{f}(x)$$

Tomorrow data, or the **test set**, will be like

$$(x_1, y_1^*), (x_2, y_2^*), \ldots, (x_n, y_n^*)$$

Here the new observations

- are n as in the training set
- have new y but old (train) x

This is a particular case which we will call **fixed-x setting** .

In general the test set is of size m and have both new x and new y

$$(x_1^*, y_1^*), (x_2^*, y_2^*), \ldots, (x_m^*, y_m^*)$$

Mean squared error

For the yesterday data (training data), we can compute the **mean squared error** (MSE)

$$\mathrm{MSE}_{\mathrm{Tr}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2$$

However, we would like to have a good performance on the test MSE

$$\mathrm{MSE}_{\mathrm{Te}} = \frac{1}{n}\sum_{i=1}^{n}(y_i^* - \hat{f}(x_i))^2$$

Polynomial regression

A reasonable choice consists of using polynomial regression (of degree d)

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \ldots + \beta_d x^d$$

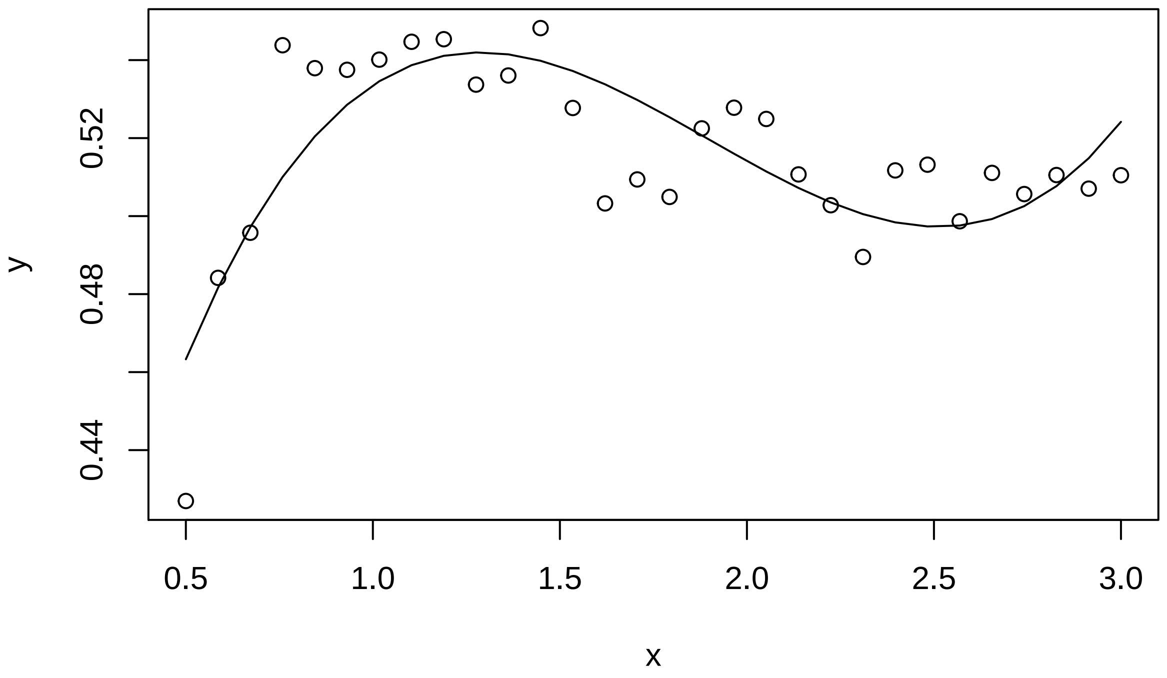
If we have no information to guide us in choosing the degree d of the polynomial, we first consider all possible degrees from 0 to $n-1$, thereby introducing $p = d+1$ parameters ranging from 1 to n, in addition to σ.

Let's try a 3rd degree polynomial model ($d = 3$)

```r
# create test set
test = data.frame(x=train$x, y=df$y.tomorrow)

# 3rd degree polynomial regression fit
fit <- lm( y ~ poly(x, degree=3), train)
yhat <- predict(fit, newdata=test)

# plot
plot( y ~ x , train)
lines( yhat ~ x, train)
```



Is this a good fit? We can compute $\mathrm{MSE}_{\mathrm{Tr}}$ for the 3rd degree polynomial model.

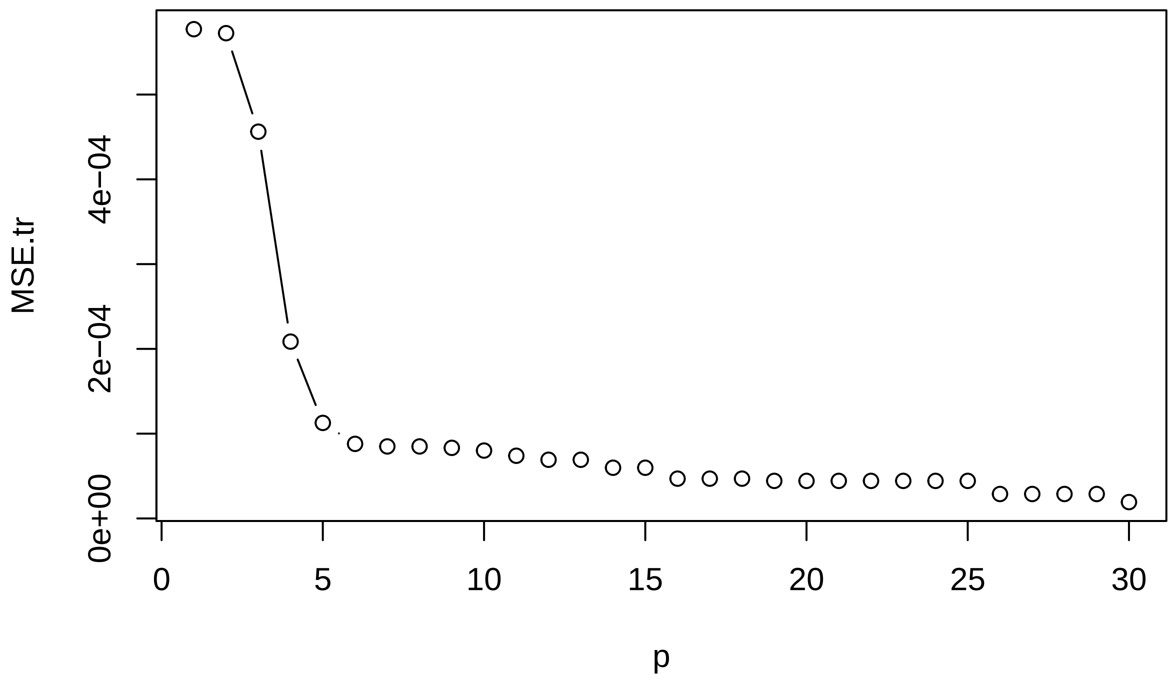
```r
# compute MSE.tr for the 3rd degree polynomial fit
MSE.tr <- mean( (train$y - yhat)^2 )
MSE.tr
```

```
[1] 0.0002085353
```

Now we compute $\mathrm{MSE}_{\mathrm{Tr}}$ for all possible degrees from 0 to $n-1$.

```r
n <- nrow(train)
ds = 0:(n-1)
ps = ds + 1
# function to fit polynomial model of degree d
fun <- function(d) if (d==0) lm(y~1, train) else lm(y~poly(x,degree=d, raw=T), train)
fits <- sapply(ds, fun)

# compute MSE.tr for all degrees
MSEs.tr <- unlist( lapply(fits, deviance) )/n
plot(ps, MSEs.tr, type="b", xlab="p", ylab="MSE.tr")
```



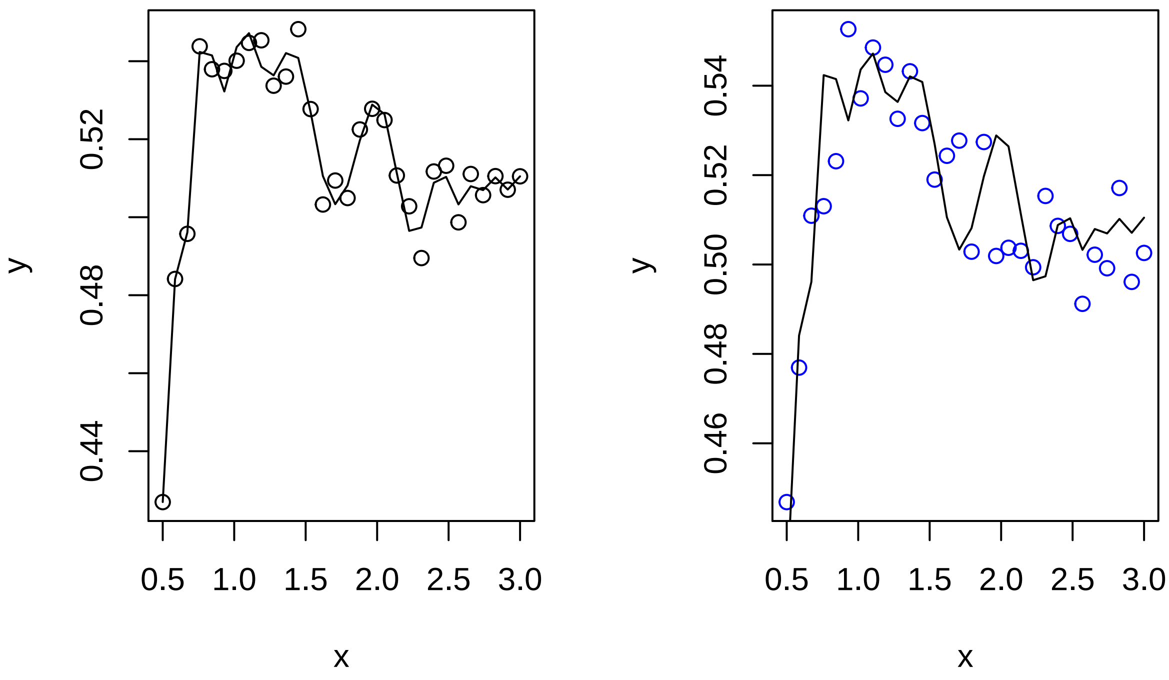
The figure shows that the polynomial fit measured by MSE_{Tr} seems to improve as p increases. A special case exists when $p = n$, corresponding to a polynomial that exactly interpolates the observed data, with $\text{MSE}_{\text{Tr}} = 0$.

Overfitting

As already mentioned, we need to use an estimate of $f(x)$ to predict values of y for new data $y_i^*, i = 1, \dots, n$ produced by the same generating mechanism, but these will become available tomorrow.

We now evaluate the quality of the prediction using yesterday's fit of the polynomials for the new y_i^*, as if we could obtain tomorrow's data today. Let's try with the 20th degree polynomial:

```r
# 20th degree polynomial fit
fit <- lm( y ~ poly(x, degree=20), train)
yhat <- predict(fit, newdata=test)
# plots
op <- par(mfrow = c(1, 2))
plot( y ~ x , train)
lines(yhat ~ x, train)
plot( y ~ x , test, col=4)
lines(yhat ~ x, train)
```

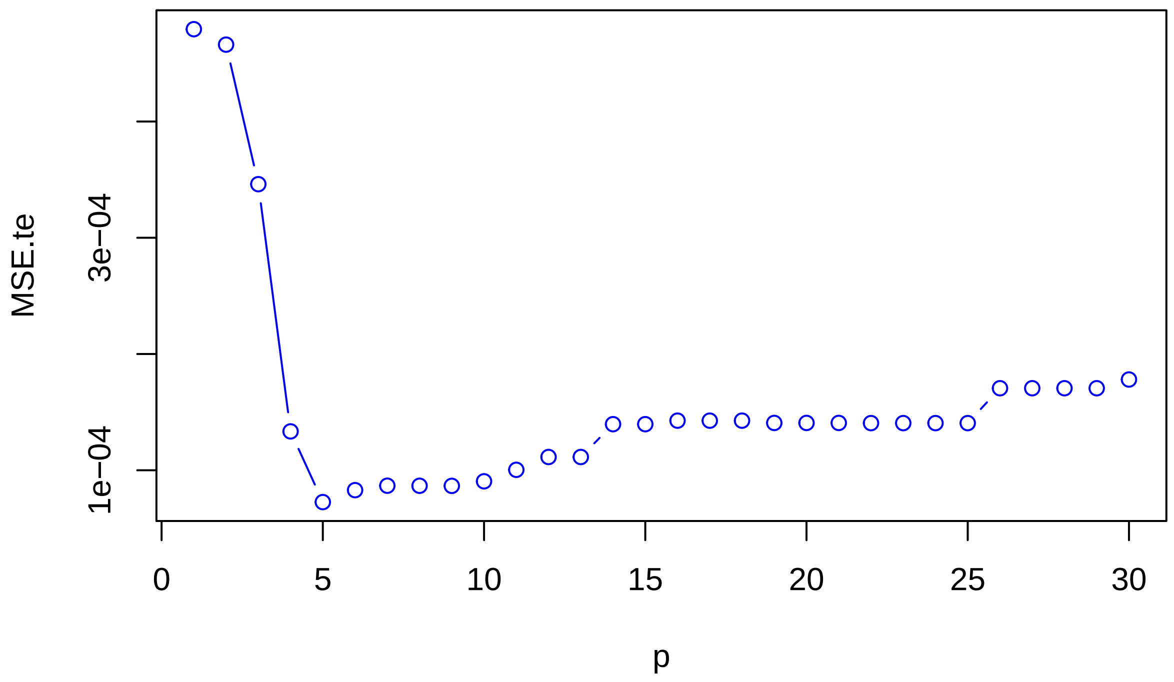


```r
par(op)
```

The figure shows yesterday and tomorrow's data with the predictions from the 20th degree polynomial. It is noteworthy that this high-degree polynomial fits very well the old points but fluctuate and no longer fit the new points. Here the model **overfits** the training data.

Now we compute $\mathrm{MSE}_{\mathrm{Te}}$ for all possible degrees from 0 to $n-1$.

```r
# compute predictions for all degrees
yhats <- lapply(fits, predict)
# compute MSE.te for all degrees
MSEs.te <- unlist(lapply(yhats,
           function(yhat) mean((test$y - yhat)^2)
           ))
# plot
plot(ps, MSEs.te, type="b", col=4, xlab="p", ylab="MSE.te")
```



```r
which.min(MSEs.te)
```

```
[1] 5
```

Here the MSE_{Te} decreases to a certain point and then increases.