Introduction to Machine Learning, Fall 2014 - Exercise session V

Rodion "rodde" Efremov
013593012

November 26, 2014

Problem 1 (3 points)

Consider the following two sets of 80 cars each: Set 'A' consists of 10 Volvos, 25 Toyotas, and 45 Audis, while set 'B' consists of 8 Volvos, 32 Toyotas, and 40 Audis. Which set do you intuitively think is more pure (that is, has lower impurity), and why? Compute the entropy, the Gini index, and the misclassification error for each of the two sets. According to these measures, which set is more pure? Could this phenomenon (conflict among the measures) happen if there were just two classes (two types of car) rather than three? Why, or why not?

To me, it seems that the set 'A' is more pure than 'B', for "intervals" in set 'A' are more "equal" than in the set 'B'.

The entropy of the set A is ($\log = \log_2$)

$$
\begin{aligned}
&-\frac{10}{80}\log\frac{10}{80} - \frac{25}{80}\log\frac{25}{80} - \frac{45}{80}\log\frac{45}{80} \\
&= \frac{1}{8}(\log 80 - \log 10) + \frac{5}{16}(\log 80 - \log 25) + \frac{9}{16}(\log 80 - \log 45) \\
&\approx 0.375 + 0.524 + 0.467 \\
&\approx 1.366.
\end{aligned}
$$

The entropy of the set B is

$$
\begin{aligned}
&-\frac{8}{80}\log\frac{8}{80} - \frac{32}{80}\log\frac{32}{80} - \frac{40}{80}\log\frac{40}{80} \\
&= \frac{1}{10}\log 10 + \frac{4}{10}(\log 80 - \log 32) + \frac{1}{2}\log 2 \\
&\approx 0.332 + 0.529 + 0.5 \\
&\approx 1.361.
\end{aligned}
$$

The Gini index of the set A is

$$
\begin{aligned}
&1 - \left(\frac{10}{80}\right)^2 - \left(\frac{25}{80}\right)^2 - \left(\frac{45}{80}\right)^2 \\
&= 1 - (1/8)^2 - (5/16)^2 - (9/16)^2 \\
&\approx 1 - 0.016 - 0.098 - 0.316 \\
&\approx 0.570.
\end{aligned}
$$

The Gini index of the set B is

$$
\begin{aligned}
&1 - \left(\frac{8}{80}\right)^2 - \left(\frac{32}{80}\right)^2 - \left(\frac{40}{80}\right)^2 \\
&\approx 1 - 0.01 - 0.16 - 0.25 \\
&\approx 0.58.
\end{aligned}
$$

The classification error of the set A is

$$1 - \frac{45}{80} = 1 - \frac{9}{16} = 0.4375.$$

The classification error of the set B is

$$1 - \frac{40}{80} = 0.5.$$

In summary: According to the above table, the set A is slightly more pure

	Entropy	Gini index	Classification error
A	1.366	0.570	0.438
B	1.361	0.580	0.5

than B.

Suppose a set contains N elements in total, and M of them belong to one type of elements and $N - M$ to the other one. Now, all the impurity measure will attain their maximum value at around $M = \frac{N}{2}$, and all measures will be monotonically increasing on $[0, \frac{N}{2})$, and monotonically decreasing on $(\frac{N}{2}, N]$, so improvement in, say, Gini index will lead to improvement in classification error or entropy, and vice versa. (As can be seen in a figure of course slides.)

Problem 2 (3 points)

Consider a binary classification problem with the following set of attributes and attribute values:

- Air conditioner = { Working, Broken }
- Engine = { Good, Bad }

- Mileage = { High, Medium, Low }
- Rust = { Yes, No }

Suppose a rule-based classifier produces the following rule set:

- (Mileage = High) ← (Value = Low)
- (Mileage = Low) ← (Value = High)
- ((Air conditioner = Working) and (Engine = Good)) ← (Value = High)
- ((Air conditioner = Working) and (Engine = Bad)) ← (Value = Low)
- (Air conditioiner = Broken) ← (Value = Low)

Given the above, answer the following:

(a) Are the rules mutually exclusive? Yes, they are. On each record, at most one rule is triggered.

(b) Is the rule set exhaustive? Suppose that Mileage = Medium. This is not covered by two first rules, but it all boils down to the fact that Air conditioner will "catch" all the records regardless of the state of Engine. So, yes, the rule set is exhaustive.

(c) Is ordering needed for this set of rules? No, it doesn't appear that way: mileage is a good predictor of the value of a car, so it is justified to have the first two rules where they are.

(d) Do you need a default class for the rule set? No, we don't need that since the rules are exhaustive.

Problem 3 (3 points)

Given a set of N points $y_1, \ldots, y_N$, with each $y_i \in \mathbb{R}$, show that...

(a) ... the value y^* which minimizes the sum of *squared* errors, i.e.

$$y^* = \arg\min_{\hat{y}} \sum_{i=1}^{N} (y_i - \hat{y})^2$$

is given by the *mean* of the y_i, i.e. $y^* = \sum_i y_i / N$.

Suppose we have

$$\sum_{i=1}^{N} (y_i - t)^2 = \sum_{i=1}^{N} (y_i^2 - 2y_i t + t^2).$$

Now the first derivative of the above sum with respect to t is given by

$$\sum_{i=1}^{N} 2t - 2y_i,$$

and the second derivative of the same sum is given by

$$\sum_{i=1}^{N} 2 = 2N > 0.$$

The original sum is, therefore, minimized (2nd derivative > 0) at point, where the first derivative is zero, i.e.

$$\begin{aligned}
\sum_{i=1}^{N} 2t - 2y_i &= 0 \\
\sum_{i=1}^{N} t - y_i &= 0 \\
Nt &= \sum_{i=1}^{N} y_i \\
t &= \frac{\sum_{i=1}^{N} y_i}{N},
\end{aligned}$$

which proves that the mean of the y_i minimizes the sum of squared errors.

(b) ... the value y^* which minimizes the sum of *absolute* errors, i.e.

$$y^* = \arg\min_{\hat{y}} \sum_{i=1}^{N} |y_i - \hat{y}|$$

is given by the *median* of the y_i. [Hint for part (b): Break the sum into parts corresponding to *pairs* of observations, pairing the smallest with the largest point, the second-smallest with the second-largest point, etc.]

Let us rephrase the sum. If N is even, we have

$$\sum_{i=1}^{N} |y_i - \hat{y}| = \sum_{i=1}^{N/2} |y_i - \hat{y}| + |y_{N+1-i} - \hat{y}|$$

Problem 4 (15 points)

In this problem, we will test linear regression on a simple synthetic dataset. We will use the following polynomial as the underlying target function

$$y = f(x) = 2 + x - 0.5x^2.$$

First, randomly sample 30 points x_i from the uniform distribution on the interval $[-3, 3]$. Then, randomly generate the y_i using

$$y_i = f(x_i) + n_i,$$

where f is as defined above, and the n_i are i.i.d. normal random variables with zero mean and standard deviation 0.4. The resulting 30 pairs (x_i, y_i) is your data set for this exercise.

(a) First, let's fit polynomials of order 0 to 10 to this dataset using linear regression, minimizing the sum of squares error. That is, fit functions of the form

$$\hat{y} = \sum_{p=0}^{K} w_p x^p$$

with $K = 0, \ldots, 10$ to the data. For instance, for $K = 4$ the polynomial to fit is

$$\hat{y} = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4.$$

For each of the 11 values of K, produce a separate plot showing the datapoints (x_i, y_i) and the fitted polynomial. (Plot the polynomial as a curve, in the full interval $[-3, 3]$, overlayed on the scatterplot of the points.) You should see that as the order of the polynomial K increases, the curve comes closer and closer to fitting all the datapoints. For each value of K, calculate the coefficient of determination R^2 and comment on the behaviour of this measure as a function of K.

First of all, run a declaration of R^2 stuff:

```
run('rsquared.m');
```

which is defined as

```
function [r] = rsquared(xs, ys)
  ssres = 0;
  sstot = 0;

  mean_x = mean(xs);

  for i = 1:length(xs)
      sstot += (xs(i) - mean_x)^2;
      ssres += (xs(i) - ys(i))^2;
  end

  r = 1 - ssres / sstot;
end
```

As to generate the 30 x_i, let us type

```
xs = unifrnd(-3, 3, 1, 30);
```

Next, let us declare the function f:

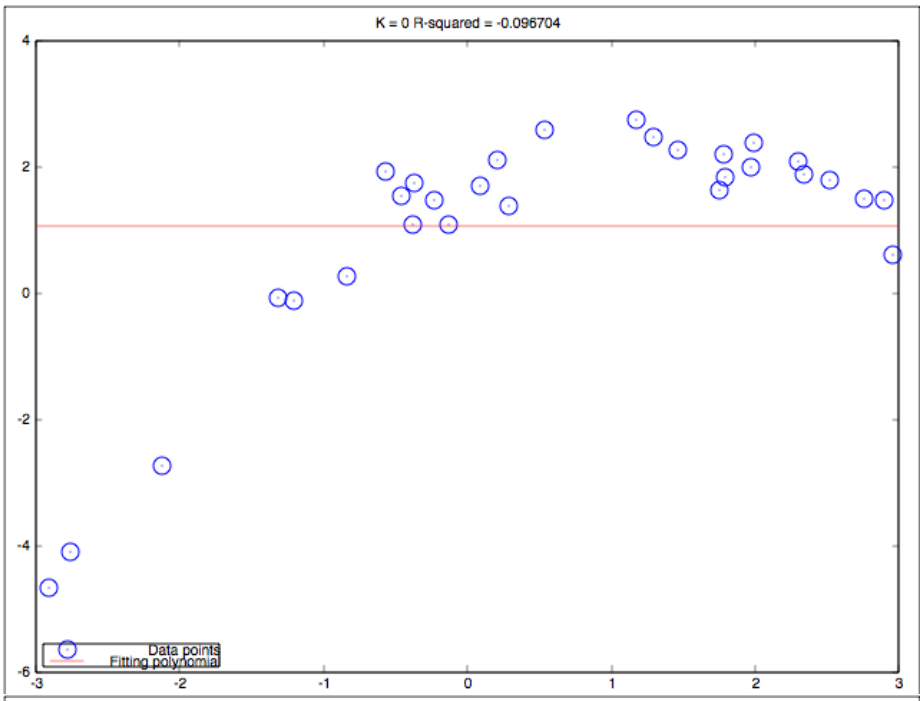
```
f = @(x) 2 + x - 0.5 * x^2
```

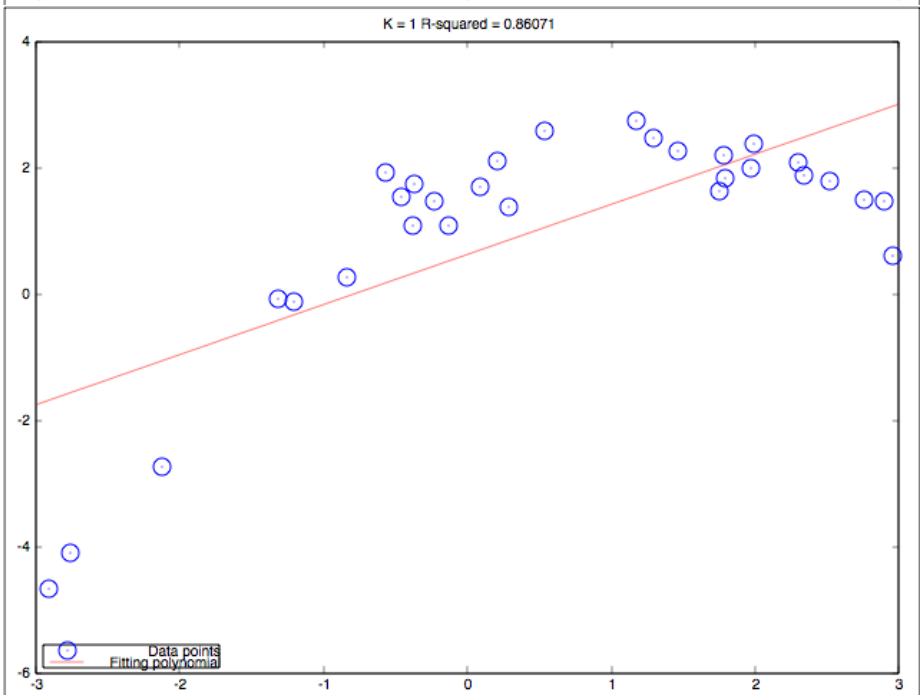
Create the y_i vector:

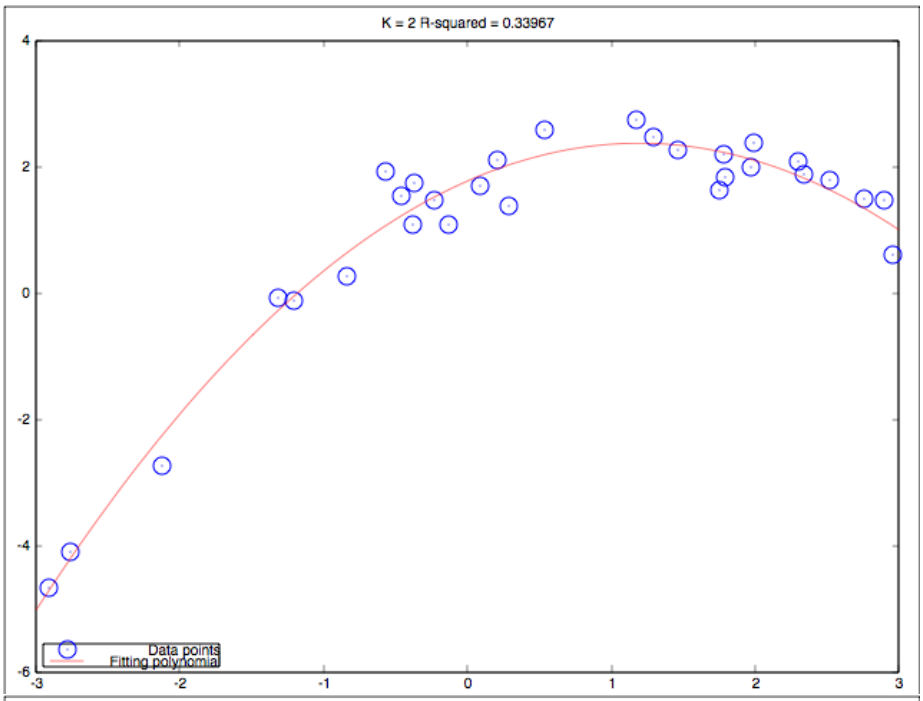
```
ys = zeros(1, 30);
for i = 1:30
  ys(i) = f(xs(i)) + normrnd(0, 0.4);
end
```

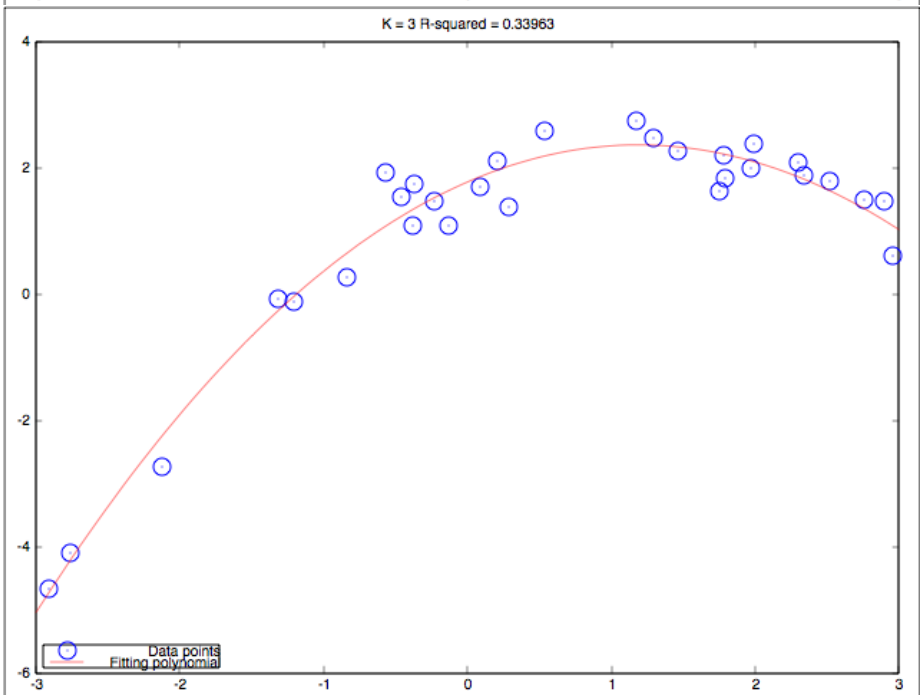
Now, data is here. Next

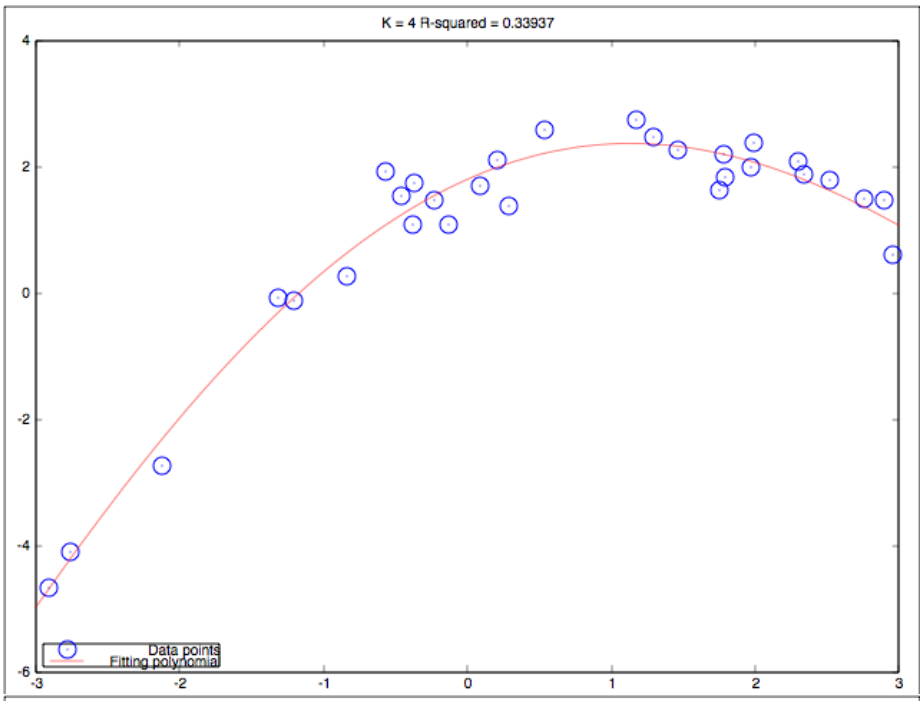
```
for K = 0:10
  figure;
  plot(xs, ys, 'o');
  hold on;
  p = polyfit(xs, ys, K);
  xt = linspace(-3, 3);
  fs = polyval(p, xt);
  fs2 = polyval(p, xs);
  plot(xt, fs, 'r-');
  title(["K = ", num2str(K), " R-squared = ", num2str(rsquared(xs, fs2))]);
  legend('Data points', 'Fitting polynomial', 'location', 'southwest');
end
```

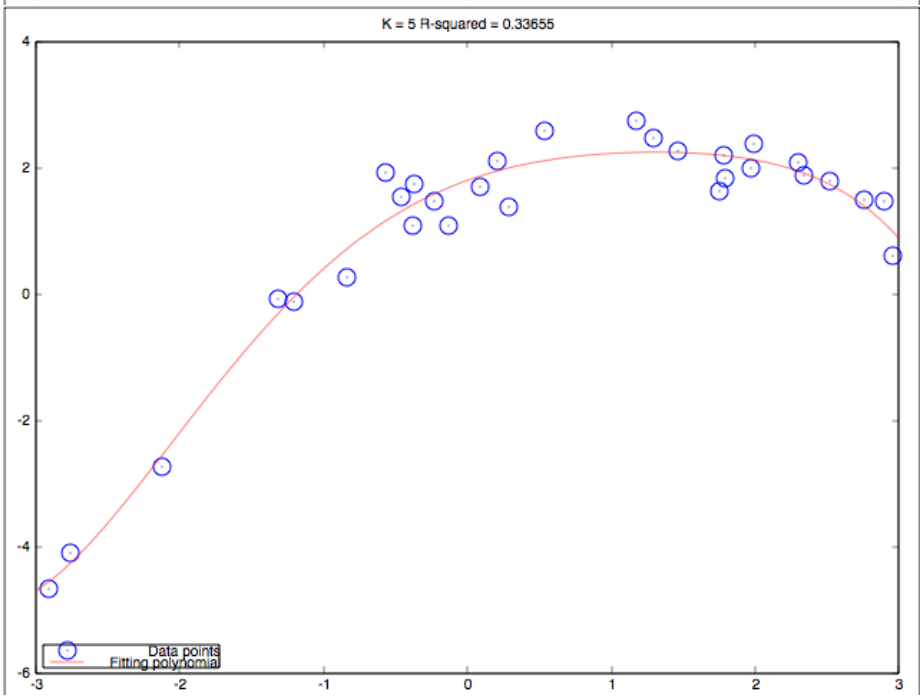

K = 0 R-squared = -0.096704
Data points
Fitting polynomial

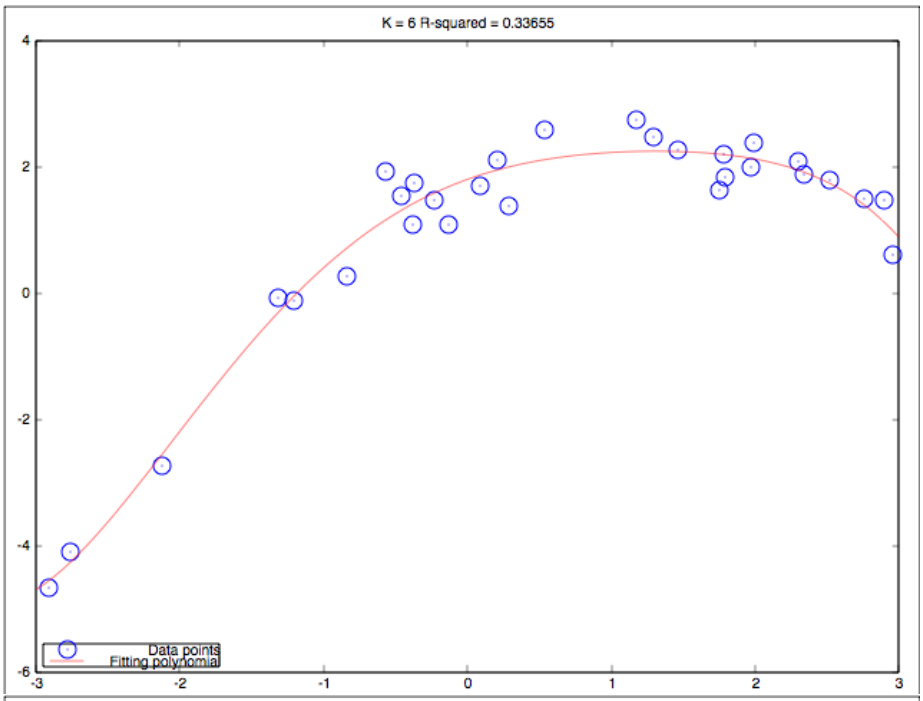

K = 1 R-squared = 0.86071
Data points
Fitting polynomial

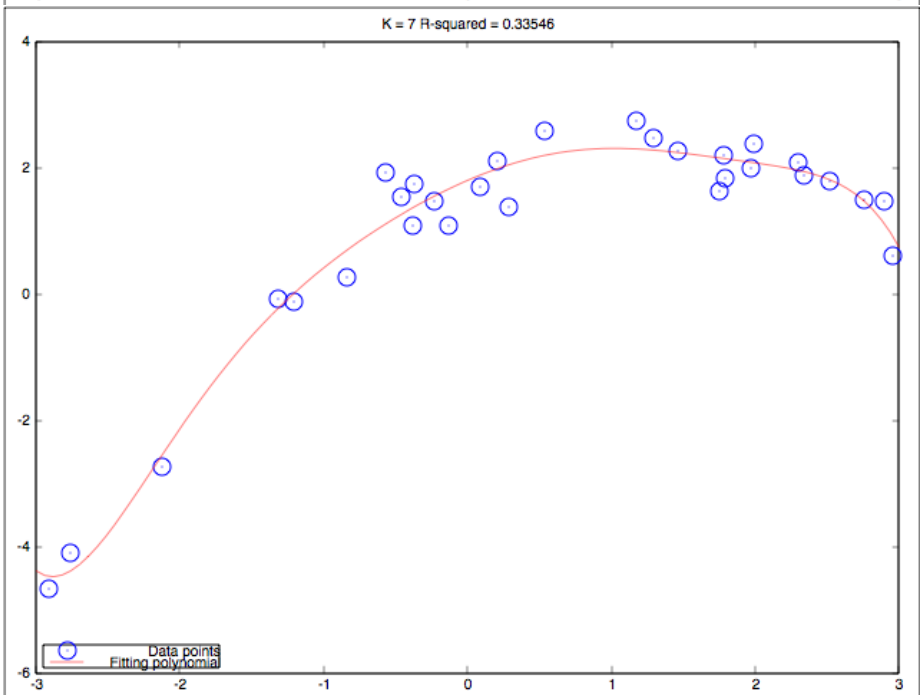

K = 2 R-squared = 0.33967
Data points
Fitting polynomial

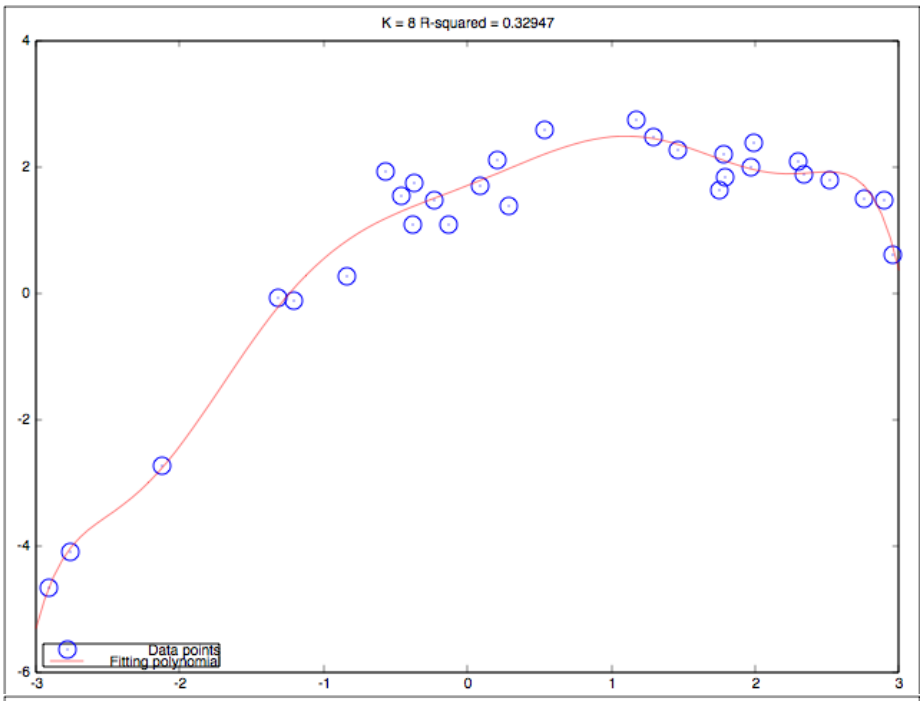

K = 3 R-squared = 0.33963
Data points
Fitting polynomial

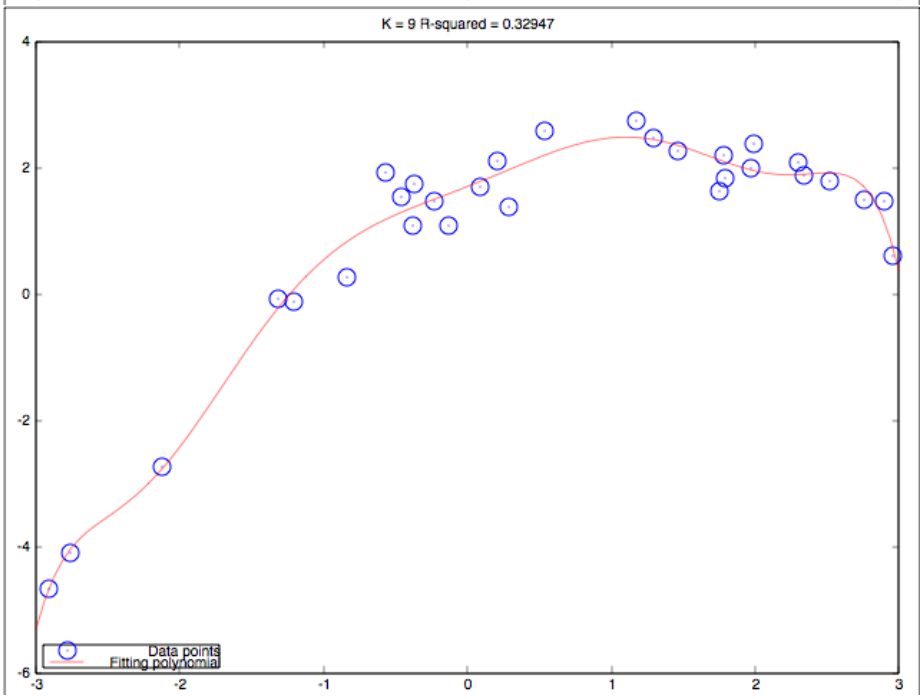

K = 4 R-squared = 0.33937
Data points
Fitting polynomial

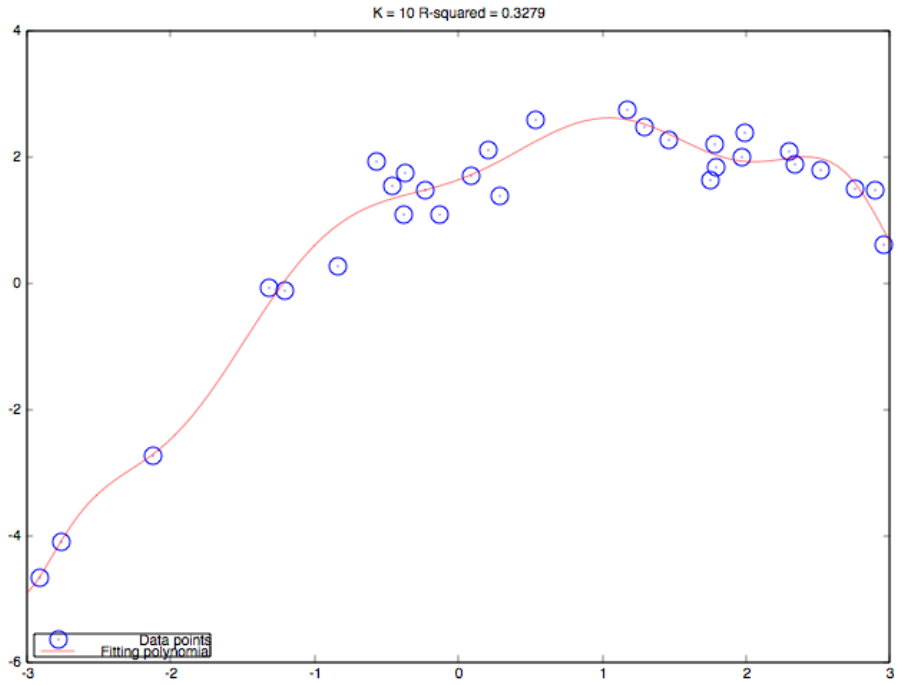

K = 5 R-squared = 0.33655
Data points
Fitting polynomial


K = 6 R-squared = 0.33655
Data points
Fitting polynomial


K = 7 R-squared = 0.33546
Data points
Fitting polynomial


K = 8 R-squared = 0.32947
Data points
Fitting polynomial


K = 9 R-squared = 0.32947
Data points
Fitting polynomial



As can be seen from the images above, the R^2 attains its maximum value at $K = 1$, after which it is strictly decreasing, and it seems like its rate of "growth" is similar to $C - \log K$, for some C.

(b) Finally, let's test model selection based on 10-fold cross-validation. That is, divide the dataset into 10 equal-sized subsets (i.e. 3 datapoints in each subset), and, for each value of $K = 0, \ldots, 11$ and each data subset $j = 1, \ldots, 10$, use all the data except the data in subset j to fit the polynomial of order K, and compute the resulting sum of squared errors on subset j. For each value of K, sum together the errors coming from the different j. Plot these results with K on the horizontal axis, and the sum of squared errors on the vertical axis. How does this function behave? Does the cross-validated error improve with increasing K? Which K gives the minimum error, and what are the corresponding estimated coefficients? Compare them to the true coefficients used in $f(x)$ to generate the data.

Now, let us split the data:

```
xsplit = zeros(10, 3);
ysplit = zeros(10, 3);
for i = 1:10
  xsplit(i,:) = xs( 3 * (i - 1) + 1 : (i - 1) * 3 + 3);
  ysplit(i,:) = ys( 3 * (i - 1) + 1 : (i - 1) * 3 + 3);
end
```

The rest is given by:

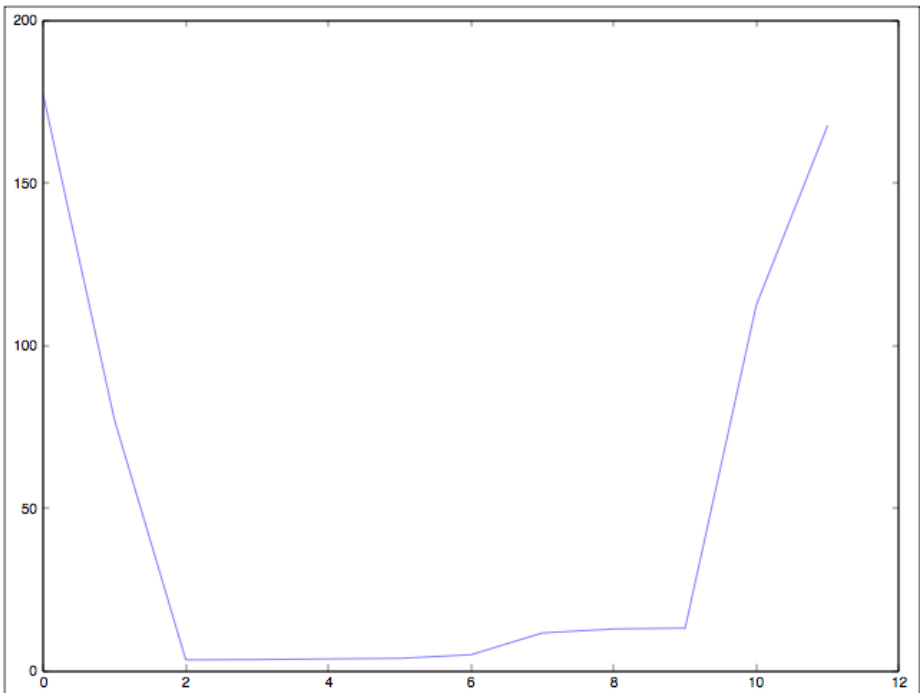
```
# For the error sums.
errs = zeros(1, 12);

# Stored polynomial coefficients.
coefs = zeros(12, 12);

# For each value of K, do...
for K = 0:11
    printf("--- K = %d.\n", K);
    # For each subset, do...
    for j = 1:10
        # Copy the data points.
        xdata = xs;
        ydata = ys;
        # Obtain the jth subset.
        xjth = xdata(3 * (j - 1) + 1 : 3 * (j - 1) + 3);
        yjth = ydata(3 * (j - 1) + 1 : 3 * (j - 1) + 3);
        # Remove the jth subset.
        xdata(3 * (j - 1) + 1 : 3 * (j - 1) + 3) = [];
        ydata(3 * (j - 1) + 1 : 3 * (j - 1) + 3) = [];
        # Fit a polynomial of degree K over the datapoints.
        p = polyfit(xdata, ydata, K);
        # Add the coefficients.
        for kk = 0:K
            coefs(K + 1, kk + 1) += p(kk + 1) / 12;
        end
        # Evaluate the jth subset.
        fjth = polyval(p, xjth);
        err = sum_of_squared_errors(fjth, yjth);
        printf('Sum of squared errors on %dth subset is %f.\n', j, err);
        errs(K + 1) += err;
    end
    # Print the error sum.
    printf("-- Error sum is %f. ", errs(K + 1));
    # Print the coefficients of the polynomial.
    printf("Coefficients: ");
    for kk = 0:K
        printf("%f ", coefs(K + 1,kk + 1));
    end
    printf("\n");
end

# Prepare to plot K-to-error graph.
Ks = zeros(1, 12);
```

```
for K = 0:11
    Ks(K + 1) = K;
end
figure
plot(Ks, errs);
end
```



It can be seen that the error function decreases rapidly from $K = 0$ to $K = 2$, where it approximately attains the minimum value, which carried over to $K = 4$, after which it start to grow towards higher values of K until it reaches $K = 9$, where things go out of control. (Depends on random number generator: sometimes values on the right side of plot do not "blow out.")

Increasing the K does not pay off after $K = 4$. The sums of squared errors is minimized at $K = 2$, where the average coefficients over all js are $1.76, 0.841, -0.452$, which resemble the coefficients $2, 1, -0.5$.