Additive Term Models (atm 0.1.0)
VERY ROUGH DRAFT

Miss Charlotte Maia

Statistician (Portfolio Management and Statistical Programming)

August 20, 2009

Abstract

The atm package is an R package for creating additive models with semiparametric predictors. It emphasizes term objects, especially the implementation of a term class hierarchy, and the interpretation and evaluation of term estimates as functions of explanatories. Typically, linear and smooth terms would be treated somewhat differently, however by creating a term superclass, and then creating linear and smooth term subclasses, we are able to unify much of their properties. The linear class is further extended by categorical, linear and interaction term classes. In order to support interpretation, linear terms generally correspond to a single variable, and (especially in the case of categorical terms) may contain multiple parameters each. Likewise, smooth terms use series-based local polynomial smoothing.

WARNING

This package is new. It should be regarded as very unstable, and has had very limited testing. It is not *yet* recommended for use in an applied context.

1 Introduction

An extremely common notion in statistics is the "linear model". Perhaps the most important part of the linear model is a linear predictor, which we will define as a function of one of more explanatories. One way to express a linear predictor is as the sum of linear terms, where roughly speaking, each term is the product of a parameter and a variable. Hence, we can also describe a linear model as an additive model, and a linear predictor as an additive predictor. Noting that for our purposes we still regard the model as additive, if it contains interactions.

However, a linear model is only a special case of an additive model. Additive models also can include smooth and spline terms. The theory used to fit an additive model with linear, smooth or other classes of terms is relatively straightforward, however creating a full featured implementation, that is not overly restrictive, is not so straightforward.

If we consider some of the existing frameworks for this purpose, they do indeed restrict us. Hence the primary goal of the atm package is to provide a framework for additive models with semiparametric predictors, that is relatively unrestrictive. This is achieved via an object oriented approach, especially a sophisticated term class hierarchy, that is discussed further in the next section. Rather than hybridise, we generalise.

The second major goal of the package, is to treat term estimates as functions of explanatories. The author is interested in applications of additive models to optimisation (e.g. determining the path of steepest ascent) and forecasting. Whilst there are different views on optimisation and forecasting, she believes it is imperative to look at functions of variables, both at a mathematical level and at a visual level. This contrasts to typical implementations (especially for linear terms), where term estimates are essentially parameters. If we want to say, plot the effect of a categorical variable with ten levels (after adjusting for other variables), this would be a lot more work than it should be. Note that this makes sense. Usually in statistics, one is interested in parameters, and the inference for such parameters.

In order to achieve this goal, the linear terms in this package, generally correspond to one variable, however (especially in the case of categorical terms) may contain multiple parameters. We can visualise the effect of the variable simply by typing plot (catterm), and evaluate a polynomial term simply by evaluate (polyterm, xvalue).

In addition the goals mentioned above, the author is experimenting with methods to make the models extremely robust to missing values in the explanatories. Presently this has not been successful, and any data used by terms must be clean. Further experiments are being performed to generalise terms by specifying an estimator.

There are a few issues with this package that are worth noting:

1. The data must be clean. Incorrect output is produced otherwise. This problem should hopefully be fixed soon.

2. The algorithm may fail to converge. Either start with a regular linear model, or use manual forward selection.

3. Smooth terms have no inference.

4. Linear terms ignore weights in their inference.

5. The inference is unreliable altogether.

6. Some features haven't been tested at all.

This isn't as bad as it sounds. Remember it is version 0.1.0.

Note that the atm package is built on top of the oosp package. Also note the term class indirectly extends the environment class, hence term objects require some special consideration. This might be discussed further in the next release. Although here is issue to be aware of.

Two compenv objects with same "value".

```
> e1 = e2 = compenv (x=0)
> e1$x
[1] 0
> e2$x
[1] 0
```

If we change one, it changes both (actually they are the same thing). Both atm objects and term objects behave the same way.

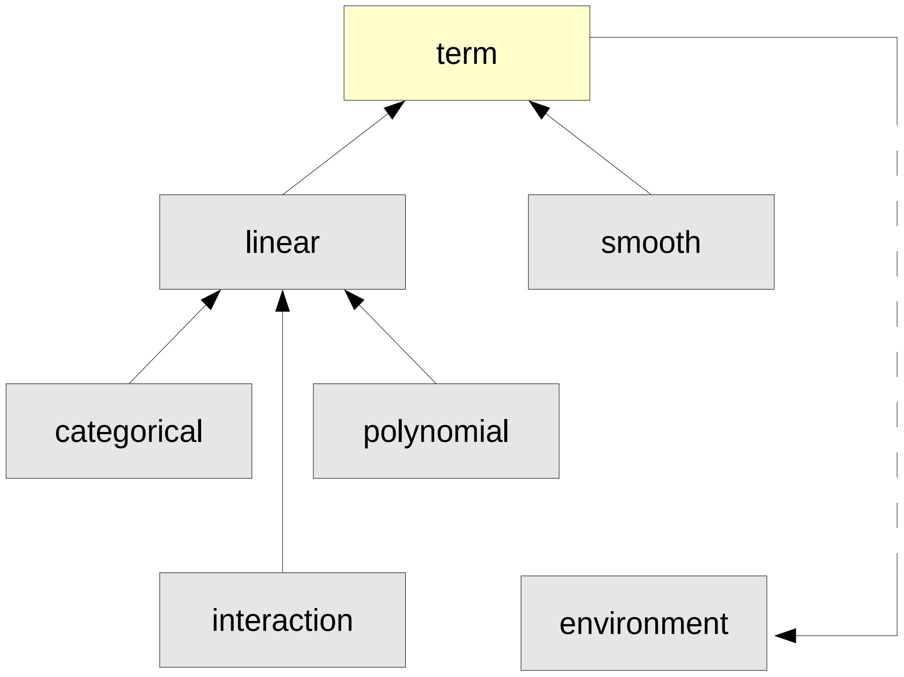
```
> e1$x = 1:10
> e1$x
 [1]  1  2  3  4  5  6  7  8  9 10
> e2$x
 [1]  1  2  3  4  5  6  7  8  9 10
```

2 Term Class Hierarchy

The term class extends the termenv class, which in turn extends the compenv class from the oosp package. These objects are environments.

In principle the term class is abstract, meaning that one can't create an instance of it. However, due to the liberal nature of S3, we still can if we want, however the user should not attempt this.

The subclasses have already been discussed. Following is a somewhat rushed diagram.



It should be noted that terms also have components, such as term design, term estimator and term estimate components. It is possible to change the estimators, however this is currently a bit rough it will be discussed further if and when the process is refined.

3 Linear Additive Models

Firstly, it is important to re-iterate, that linear terms in this package, can contain multiple parameters. Also, just so that there is no misunderstanding "linear" essentially refers to linearity in parameters, where as "polynomial" which we also consider to be linear, is a polynomial in a variable.

Categorical, polynomial and interaction terms, add little to linear terms in the way of functionality. They are mainly there to make it easier to create a linear term. Historically one would work out a design matrix. With R's lm function, obviously you don't do this, rather you provide a formula object. From personal experience, formula objects are good for simple models created at the command line. However for more exotic models, or models created inside another algorithm, they can become awkward. Hence the approach here is different, a linear term contains what we call key functions (noting the author is not aware of any standard word). We define a key function to be a function that maps what we call a real world variable, to a model world variable. So a polynomial $\hat{\theta}_1 x + \hat{\theta}_2 x^2$, can be thought of as an expression with one real world variable, two model world variables, and two key functions x and x^2. We must specify key functions, if we wish to create non-standard linear terms. However for standard linear terms, we just use one of the linear subclasses, e.g. the categorical term constructor creates the key functions for us.

Also note that categorical terms contain one parameter for each variable level (i.e. there is a baseline estimate), and that non-standard linear terms, such as $\log(x)$, are only permitted if the function of x, produces a valid value, for every valid realised value of x. So if an explanatory x, contains zeros, then we can not use $\log(x)$. The default key function for a pure linear term and for a polynomial term is x. Although see the following note on polynomials.

Now let us consider an example. Here we have a simulated dataset.

```
> data (sample)
> sample [1:4,]
         x1        x2 x3         x4         y
1  3.3333333  8.787879  A -2.1212121  77.07694
2 -0.7070707  8.383838  B -9.7979798  53.52644
3 -1.5151515 -4.343434  C  0.7070707  58.49027
4  2.5252525  7.171717  D  8.7878788 117.44342
> attach (sample)
```

We can create a term object in two ways. If we wish to fit an atm model, then we omit y. However if we wish to fit a single term in isolation, we include y. The term will automatically fit itself if y is provided. In the case of a polynomial term, the presence or absence of y, determines whether or not an intercept parameter is included by default.

So these are the roughly the same and do not fit themselves (plus there no intercept).

```
> t = linear (x1)
> t = poly (x1)
```

These are also roughly the same, however do fit themselves (plus there is also no intercept).

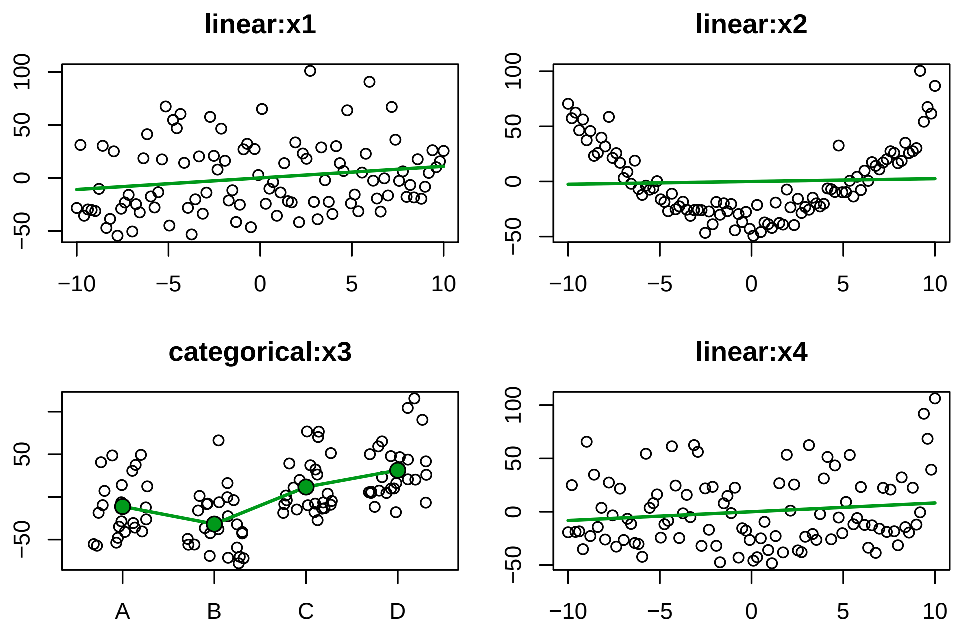
```
> t = linear (x1, y)
> t = linear (x1)
> fit (t, y)
```

These are not the same (the polynomial will include an intercept, unless the argument intercept=FALSE is included).

```
> t = linear (x1, y)
> t = poly (x1, y)
```

Fitting a single linear term is not all that interesting. Let us now consider an atm model. All of the variables are continuous, except for x3. If we make an assumption that all variables effect the response (unwise in practice), then perhaps the simplest reasonable model is the following:

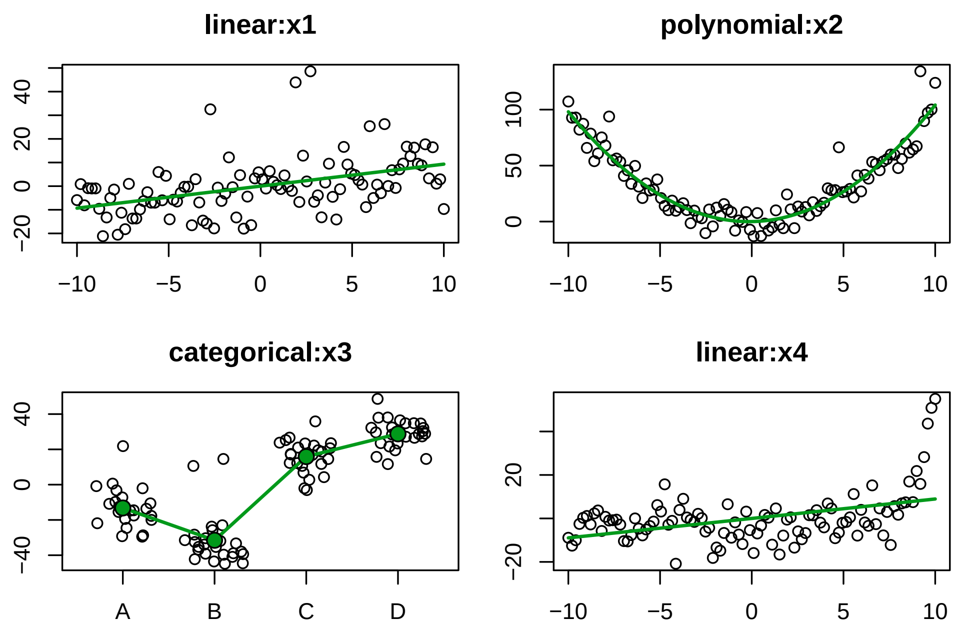
```
> t1 = linear (x1)
> t2 = linear (x2)
> t3 = categorical (x3)
> t4 = linear (x4)
> m = atm (y, t1, t2, t3, t4)


> plot (m)
```



There is a possible polynomial effect that has not been captured so we might try something like this.

```
> t2 = polynomial (x2, deg=2)
> m = atm (y, t1, t2, t3, t4)
```

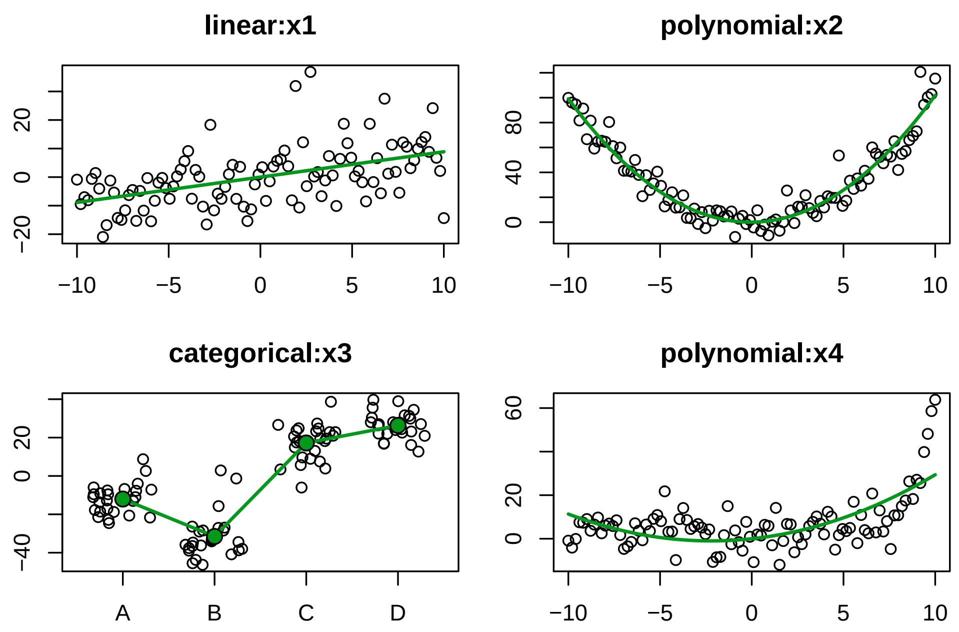
```
> plot (m)
```



Repeating for the other term.

```
> t4 = polynomial (x4, deg=2)
> m = atm (y, t1, t2, t3, t4)
```

```
> plot (m)
```



Obviously that isn't completely successful. We might try a third or fourth degree polynomial, if we wanted to be adventurous. However we will consider a smooth term in the next section.

We can evaluate terms using the evaluate function.

```
> evaluate (t3, "D")
[1] 26.3353
```

To evaluate the entire atm model, we use a list of equal length vectors. Interactions are slightly different. However they are still being tested and are not discussed here.

```
> evaluate (m, list (0, 0, "D", 0) )
[1] 44.21896
```

We can print or produce summary output for both the atm model and term objects, in the same way as lm objects, except that for terms, summary output requires us to provide a y argument (either the response, or partial residuals). It may be easier to use the atm version and specify a which argument instead.

Also re-iterating, at present, the inference is unreliable altogether. This should be improved in future releases.

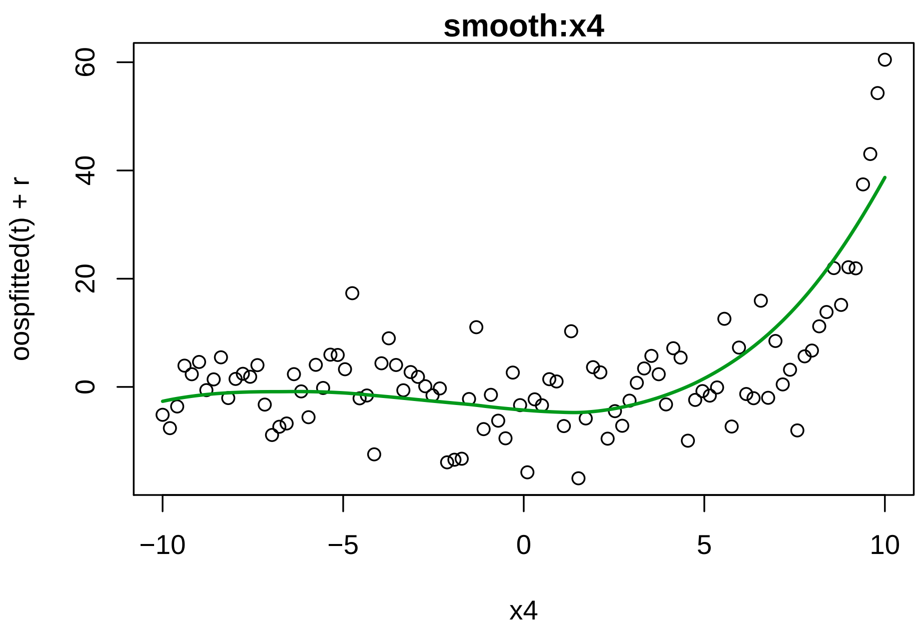
```
> summary (m, 3)
$label
[1] "categorical:x3"

$weighted
```

```
[1] FALSE

$e
  labs        th       se         tv         pv stars
1    A -12.01964 16.18091 -0.7428284 0.45939947
2    B -31.52316 16.18091 -1.9481695 0.05431517     .
3    C  17.20750 16.18091  1.0634442 0.29024826
4    D  26.33530 16.18091  1.6275536 0.10689741

$ft
$ft$fstat
[1] -23.96883

$ft$pvalue
[1] 1


$gf
       mar       rsq
1 64.08408 0.5837024

attr(,"class")
[1] "summary.linear"
```

4 Smooth Terms

As promised, here is a smooth term, although not that great.

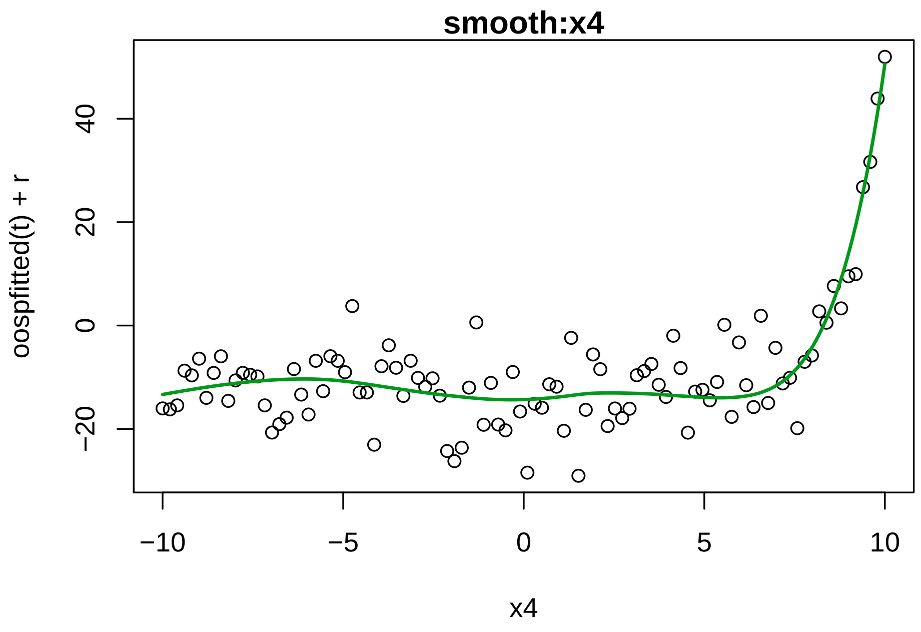
```
> t4 = smooth (x4)
> m = atm (y, t1, t2, t3, t4)


> plot (m, which=4)
```



We can try again with some different arguments. Note that the author is developing better smoothing methods, plus is considering adding a spline term (based on piecewise functions with fixed knots) to the package.

```
> t4 = smooth (x4, deg=4, refit=2)
> m = atm (y, t1, t2, t3, t4)
```

```
> plot (m, which=4)
```



Whilst we can use the evaluate function. We can also view the model as a series.

```
> t4$d$sx
 [1] -10.0000000  -9.5918367  -9.1836735  -8.7755102  -8.3673469  -7.9591837
 [7]  -7.5510204  -7.1428571  -6.7346939  -6.3265306  -5.9183673  -5.5102041
[13]  -5.1020408  -4.6938776  -4.2857143  -3.8775510  -3.4693878  -3.0612245
[19]  -2.6530612  -2.2448980  -1.8367347  -1.4285714  -1.0204082  -0.6122449
[25]  -0.2040816   0.2040816   0.6122449   1.0204082   1.4285714   1.8367347
[31]   2.2448980   2.6530612   3.0612245   3.4693878   3.8775510   4.2857143
[37]   4.6938776   5.1020408   5.5102041   5.9183673   6.3265306   6.7346939
[43]   7.1428571   7.5510204   7.9591837   8.3673469   8.7755102   9.1836735
[49]   9.5918367  10.0000000
> t4$e$sy
 [1] -13.338927 -12.833611 -12.360077 -11.909568 -11.504787 -11.156605
 [7] -10.863250 -10.625846 -10.450685 -10.353912 -10.344686 -10.448500
[13] -10.668591 -10.992463 -11.396299 -11.830851 -12.253653 -12.673168
[19] -13.061278 -13.410703 -13.718737 -14.002163 -14.223973 -14.344960
[25] -14.357011 -14.262435 -14.065662 -13.781833 -13.426117 -13.128062
[31] -13.043483 -13.051819 -13.117580 -13.225937 -13.394009 -13.596585
[37] -13.773598 -13.929948 -13.976124 -13.838974 -13.413633 -12.521530
[43] -10.935533  -8.374274  -4.505000   1.056841   8.756840  19.116245
[49]  32.781896  50.597126
```

Whilst the series is fitted using a local polynomial smoother. Evaluation is done via a spline. The actual implementation is given.

```
> evaluate.smooth
function (t, u, ...)
spline(t$d$sx, t$e$sy, xout = u)$y
```