The Kernel Trick (Continued)

There are many types of kernels out there, and with SciKit-Learn, you can even define your own. Knowing what type of kernel to use, and under what circumstances to use it, are things that you'll have to discern as a domain expert in the field you are researching. A kernel that performs amazingly for one problem might fail entirely with a different problem. Let's solidify your understanding of kernels through a few examples.

Each ball in our billiards dataset has an x-coordinate, a y-coordinate and an even-odd status. Ball 1 and 2 look like this:

```
Ball(Index),  XPos,  YPos,  Even
1,               7,     4,     0
12,              2,     3,     1
```

In order to be linearly separable, we need to get the balls into the air, a complex transformation. We can model the table-flipping as a function `f(s)`, that takes in a low-dimensional ball sample, and maps it to the higher-dimensional space once the flip has occurred:

```
f(s) = (s1*s1,  s1*s2,  s1*s3,
        s2*s1,  s2*s2,  s2*s3,
        s3*s1,  s3*s2,  s3*s3)
```

Where s1 is the sample's first feature **XPos**, s2 is the second feature **YPos**, and s3 is the third feature **Even**. This is what your billiard balls look like in the air:

```
Ball  s1*s1  s1*s2  s1*s3  s2*s1  s2*s2  s2*s3  s3*s1  s3*s2  s3*s3
1       49     28      0     28     16      0      0      0      0
12       4      6      2      6      9      3      2      3      1
```

Things are starting to get hairy and you haven't even slipped the paper between the balls! To do that, you'll have to see how similar each ball vector is. You can use a scalar dot product do to that, defined as:

```
<f(a), f(b)>  =  (a.Feature_1 * b.Feature_1)  +
                 (a.Feature_2 * b.Feature_2)  +
                 ...                          +
                 (a.Feature_n * b.Feature_n)
```

Where `a` and `b` are balls. Calculating the dot product between Ball #1 and Ball #12 once they're in the air (in the high-dimensional feature space), the calculation would look like this:

```
<f(Ball1), f(Ball12)>  =  49*4  +  28*6  +  0*2  +  28*6  +  16*9  +  0*3  +  0*2  +
0*3  +  0*1
<f(Ball1), f(Ball12)>  =  676
```

Being a ninja, you're not going to have any of that. Rather, using your domain expertise, you derive an interesting kernel function that takes in any two ball samples (`a`, `b`) in their original, non-flipped, low-dimensional table space of (`XPos`, `YPos`, `Even`), and computes for you the same metric:

```
K(a, b)  =  (a.XPos * y.XPos  +  a.YPos * b.YPos  +  a.Even * b.Even) ** 2
```

Placing Ball #1 and Ball #12 into your kernel yields:

```
K(Ball1, Ball12)  =  (7*2  +  4*3  +  0*1) ** 2
K(Ball1, Ball12)  =  676
```

Amazing—the same metric, without all the hard work! In SciKit-Learn, the built-in kernels functions provided are essentially `K(Ball1, Ball12)`. You can also specify a user defined function if you'd like to implement your own kernel function.