PCA

December 20, 2016

1 Principal Component Analysis

PCA is a dimensionality reduction technique; it lets you distill multi-dimensional data down to fewer dimensions, selecting new dimensions that preserve variance in the data as best it can.

We're not talking about Star Trek stuff here; let's make it real - a black & white image for example, contains three dimensions of data: X position, Y position, and brightness at each point. Distilling that down to two dimensions can be useful for things like image compression and facial recognition, because it distills out the information that contributes most to the variance in the data set.

Let's do this with a simpler example: the Iris data set that comes with scikit-learn. It's just a small collection of data that has four dimensions of data for three different kinds of Iris flowers: The length and width of both the petals and sepals of many individual flowers from each species. Let's load it up and have a look:

```
In [1]: from sklearn.datasets import load_iris
        from sklearn.decomposition import PCA
        import pylab as pl
        from itertools import cycle

        iris = load_iris()

        numSamples, numFeatures = iris.data.shape
        print(numSamples)
        print(numFeatures)
        print(list(iris.target_names))
```

```
150
4
['setosa', 'versicolor', 'virginica']
```

So, this tells us our data set has 150 samples (individual flowers) in it. It has 4 dimensions - called features here, and three distinct Iris species that each flower is classified into.

While we can visualize 2 or even 3 dimensions of data pretty easily, visualizing 4D data isn't something our brains can do. So let's distill this down to 2 dimensions, and see how well it works:

```
In [2]: X = iris.data
        pca = PCA(n_components=2, whiten=True).fit(X)
        X_pca = pca.transform(X)
```

What we have done is distill our 4D data set down to 2D, by projecting it down to two orthogonal 4D vectors that make up the basis of our new 2D projection. We can see what those 4D vectors are, although it's not something you can really wrap your head around:

```
In [3]: print(pca.components_)
```

```
[[ 0.36158968 -0.08226889  0.85657211  0.35884393]
 [-0.65653988 -0.72971237  0.1757674   0.07470647]]
```

Let's see how much information we've managed to preserve:

```
In [4]: print(pca.explained_variance_ratio_)
        print(sum(pca.explained_variance_ratio_))
```

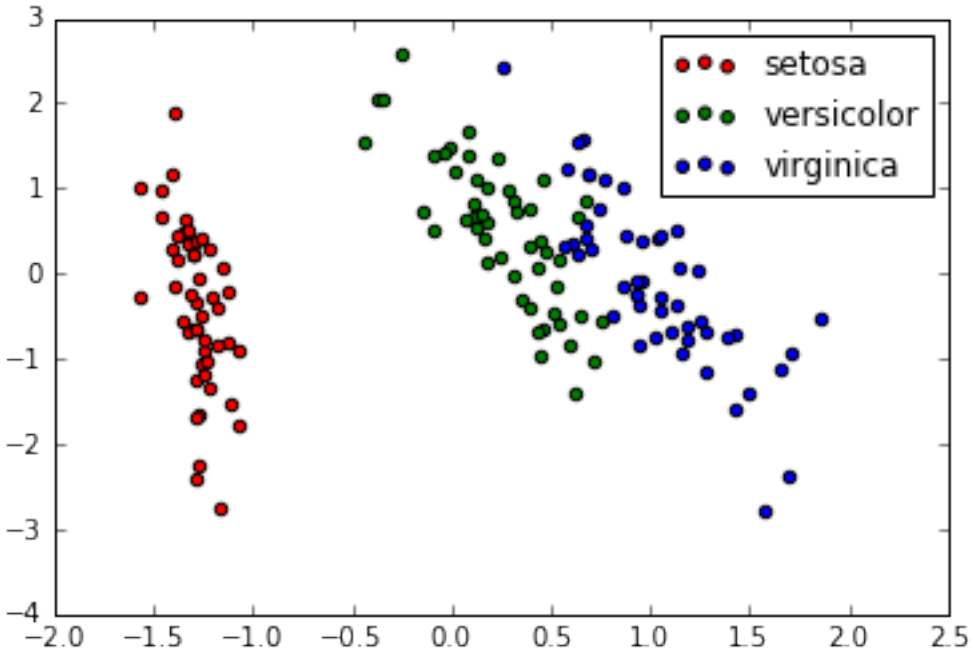
```
[ 0.92461621  0.05301557]
0.977631775025
```

That's pretty cool. Although we have thrown away two of our four dimensions, PCA has chosen the remaining two dimensions well enough that we've captured 92% of the variance in our data in a single dimension alone! The second dimension just gives us an additional 5%; altogether we've only really lost less than 3% of the variance in our data by projecting it down to two dimensions.

As promised, now that we have a 2D representation of our data, we can plot it:

```
In [5]: %matplotlib inline
        from pylab import *

        colors = cycle('rgb')
        target_ids = range(len(iris.target_names))
        pl.figure()
        for i, c, label in zip(target_ids, colors, iris.target_names):
            pl.scatter(X_pca[iris.target == i, 0], X_pca[iris.target == i, 1],
                c=c, label=label)
        pl.legend()
        pl.show()
```



You can see the three different types of Iris are still clustered pretty well. If you think about it, this probably works well because the overall size of an individual flower probably makes both the petal and sepal sizes increase by a similar amount. Although the actual numbers on this graph have no intuitive meaning, what we're probably seeing is measure of the ratio of width to height for petals and sepals - and PCA distilled our data down to that on its own.

1.1 Activity

Our results suggest we could actually distill this data down to a single dimension and still preserve most of its variance. Try it! Do a PCA down to one component, and measure the results.

```
In [ ]:
```