A PCA example with the Iris dataset

Let's apply principal component analysis to the Iris dataset. This is a 4D dataset that we're going to reduce down to 2 dimensions. We're going to see that we can actually still preserve most of the information in that dataset, even by throwing away half of the dimensions. It's pretty cool stuff, and it's pretty simple too. Let's dive in and do some principal component analysis and cure the curse of dimensionality. Go ahead and open up the `PCA.ipynb` file.

It's actually very easy to do using scikit-learn, as usual! Again, PCA is a dimensionality reduction technique. It sounds very science-fictiony, all this talk of higher dimensions. But, just to make it more concrete and real again, a common application is image compression. You can think of an image of a black and white picture, as 3 dimensions, where you have width, as your x-axis, and your y-axis of height, and each individual cell has some brightness value on a scale of 0 to 1, that is black or white, or some value in between. So, that would be 3D data; you have 2 spatial dimensions, and then a brightness and intensity dimension on top of that.

If you were to distill that down to say 2 dimensions alone, that would be a compressed image and, if you were to do that in a technique that preserved the variance in that image as well as possible, you could still reconstruct the image, without a whole lot of loss in theory. So, that's dimensionality reduction, distilled down to a practical example.

Now, we're going to use a different example here using the Iris dataset, and scikit-learn includes this. All it is is a dataset of various Iris flower measurements, and the species classification for each Iris in that dataset. And it has also, like I said before, the length and width measurement of both the petal and the sepal for each Iris specimen. So, between the length and width of the petal, and the length and width of the sepal we have 4 dimensions of feature data in our dataset.

We want to distill that down to something we can actually look at and understand, because your mind doesn't deal with 4 dimensions very well, but you can look at 2 dimensions on a piece of paper pretty easily. Let's go ahead and load that up:

```
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)
```

There's a handy dandy `load_iris()` function built into scikit-learn that will just load that up for you with no additional work; so you can just focus on the interesting part. Let's take a look at what that dataset looks like, the output of the preceding code is as follows:

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

You can see that we are extracting the shape of that dataset, which means how many data points we have in it, that is `150`, and how many features, or how many dimensions that dataset has, and that is `4`. So, we have `150` Iris specimens in our dataset, with 4 dimensions of information. Again, that is the length and width of the sepal, and the length and width of the petal, for a total of `4` features, which we can think of as `4` dimensions.

We can also print out the list of target names in this dataset, which are the classifications, and we can see that each Iris belongs to one of three different species: Setosa, Versicolor, or Virginica. That's the data that we're working with: 150 Iris specimens, classified into one of 3 species, and we have 4 features associated with each Iris.

Let's look at how easy PCA is. Even though it's a very complicated technique under the hood, doing it is just a few lines of code. We're going to assign the entire Iris dataset and we're going to call it X. We will then create a PCA model, and we're going to keep `n_components=2`, because we want 2 dimensions, that is, we're going to go from 4 to 2.

We're going to use `whiten=True`, that means that we're going to normalize all the data, and make sure that everything is nice and comparable. Normally you will want to do that to get good results. Then, we will fit the PCA model to our Iris dataset `X`. We can use that model then, to transform that dataset down to 2 dimensions. Let's go ahead and run that. It happened pretty quickly!

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

Please think about what just happened there. We actually created a PCA model to reduce 4 dimensions down to `2`, and it did that by choosing 2 4D vectors, to create hyperplanes around, to project that 4D data down to 2 dimensions. You can actually see what those 4D vectors are, those eigenvectors, by printing out the actual components of PCA. So, **PCA** stands for **Principal Component Analysis**, those principal components are the eigenvectors that we chose to define our planes about:

```
print pca.components_
```

Output to the preceding code is as follows:

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

You can actually look at those values, they're not going to mean a lot to you, because you can't really picture 4 dimensions anyway, but we did that just so you can see that it's actually doing something with principal components. So, let's evaluate our results:

```
print pca.explained_variance_ratio_
print sum(pca.explained_variance_ratio_)
```

The PCA model gives us back something called `explained_variance_ratio`. Basically, that tells you how much of the variance in the original 4D data was preserved as I reduced it down to 2 dimensions. So, let's go ahead and take a look at that:

```
[ 0.92461621  0.05301557]
0.977631775025
```

What it gives you back is actually a list of 2 items for the 2 dimensions that we preserved. This is telling me that in the first dimension I can actually preserve 92% of the variance in the data, and the second dimension only gave me an additional 5% of variance. If I sum it together, these 2 dimensions that I projected my data down into, I still preserved over 97% of the variance in the source data. We can see that 4 dimensions weren't really necessary to capture all the information in this dataset, which is pretty interesting. It's pretty cool stuff!

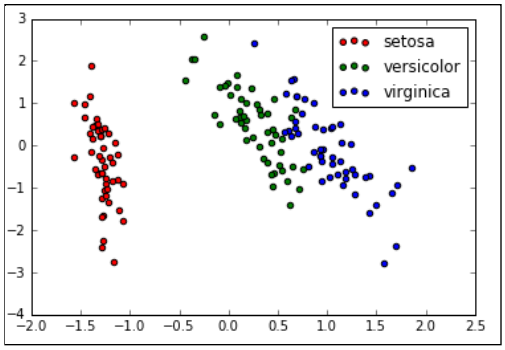
If you think about it, why do you think that might be? Well, maybe the overall size of the flower has some relationship to the species at its center. Maybe it's the ratio of length to width for the petal and the sepal. You know, some of these things probably move together in concert with each other for a given species, or for a given overall size of a flower. So, perhaps there are relationships between these 4 dimensions that PCA is extracting on its own. It's pretty cool, and pretty powerful stuff. Let's go ahead and visualize this.

The whole point of reducing this down to 2 dimensions was so that we could make a nice little 2D scatter plot of it, at least that's our objective for this little example here. So, we're going to do a little bit of Matplotlib magic here to do that. There is some sort of fancy stuff going on here that I should at least mention. So, what we're going to do is create a list of colors: red, green and blue. We're going to create a list of target IDs, so that the values 0, 1, and 2 map to the different Iris species that we have.

What we're going to do is zip all this up with the actual names of each species. The for loop will iterate through the 3 different Iris species, and as it does that, we're going to have the index for that species, a color associated with it, and the actual human-readable name for that species. We'll take one species at a time and plot it on our scatter plot just for that species with a given color and the given label. We will then add in our legend and show the results:

```
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,
        c=c, label=label)
pl.legend()
pl.show()
```

The following is what we end up with:



That is our 4D Iris data projected down to 2 dimensions. Pretty interesting stuff! You can see it still clusters together pretty nicely. You know, you have all the Virginicas sitting together, all the Versicolors sitting in the middle, and the Setosas way off on the left side. It's really hard to imagine what these actual values represent. But, the important point is, we've projected 4D data down to 2D, and in such a way that we still preserve the variance. We can still see clear delineations between these 3 species. A little bit of intermingling going on in there, it's not perfect you know. But by and large, it was pretty effective.

PREV
Principal component analysis

NEXT
Activity