Blog Post 4: Principal Component Analysis (PCA)

Before we go on to PCA, to get the flow of thought you must have read
https://codingmachinelearning.wordpress.com/2015/11/06/kaggle-handwritten-digit-recognition-part-3/

What is PCA ?

- Many say PCA is used for dimensionality reduction, and I beg to differ from that view. I feel it is not just dimensionality reduction. PCA is much more than that. It would be a terrible understatement if we said PCA is used just for dimensionality reduction.
- *PCA explains the variance in the data*. That is what PCA is actually used for. It describes the axis or dimensions along which the variance is maximum. In other words, it tells, which axis (or features) captures the most variance in the data. That is the main user of PCA. When projecting the data on a selected few principal components it results in dimensionality reduction. So we can treat dimensionality reduction as a by-product of PCA. PCA helps us identify the variance held by each axis.
- When we use PCA (as given by scikit/sklearn) it calculates the principal components and projects the data on the selected axis.

Why do we need PCA ?

PCA removes the unnecessary details in the data. For example, in the digit recognition problem, only the characters make sense. There are so many zeros which do not add sense or extra information to the data. So by discarding these axes, we can project the data on the principal axis which captures most variance.

Take the following example:

Round	Color	Fruit/Label
yes	red	apple
yes	green	mosambi

Given the features of the object (round and color) we need to classify it as fruit (apple/mosambi).We can see that Round feature really does not contribute much to the decision making process. We can remove these values. To do this in scikit, we have

CODE:

```
from sklearn.decomposition import PCA
pca=PCA(n_components=<the number of axis>)
transformed_training_data=pca.fit_transform(training_data)
clf=SVC()
```

The above piece of code projects the data on the principal axis, now that we have data on a different set of axis, we can now train the SVM classifier as described in the previous posts.

Now we also have to transform the test data also to the same axis. This is because the training data is trained on a different axis than original. But test set has the original axis , so it need to be projected on the PCA of the Training set to get it to a standard comparable form. When projected on the PC of training_data we can then predict output on the new test set.

The code given above is a working example of PCA. It helps in capturing variance in the data. Based on the variance captured, we can use dimensionality reduction as a step towards filtering unwanted non-contributing axis in the data set.

```
new_test_data=pca.transform(test_data)

output=clf.predict(new_test_data)
```

Results:

Now if we apply PCA to the digit data set, and use SVM on the modified data we can see that:

- The code runs in less than 3 minutes (which is at least 10x faster)
- Accuracy shoots up to 98% from 74%

The reason for the speedup is reduction in dimension. From 784 dimension space we have come down to 35 dimension space (we can set the number of axis based on the variance captured)

Conclusion:

- PCA explains variance in data
- Also used as a tool in dimensionality reduction
- Number of principal axis is decided by looking at the explained variance ratio of each axis to the data

Next Post:

I hope to make it a point to post a blog every *Thursday*. The entire code for the digit recognition problem is uploaded on my github account.

Github link: https://github.com/vsuriya93/coding-machine-learning

In the next post, we will look into using SVD in dimensionality reduction in context of representing word vectors. Stay tuned.