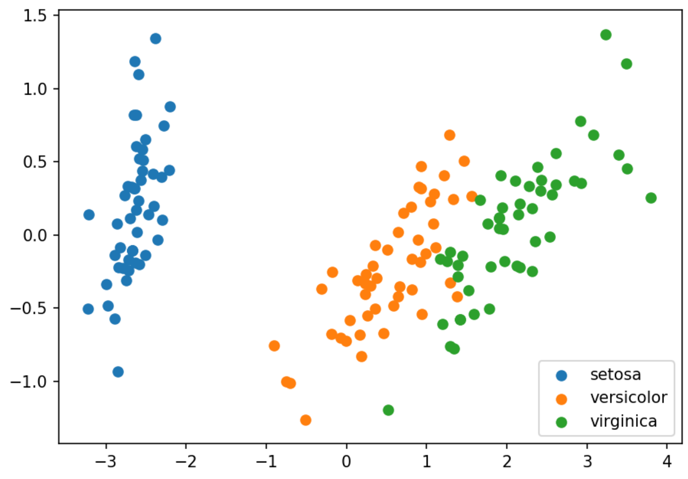
PCA in Python with SciKit Learn

stamfordresearch.com/pca-in-python-with-scikit-learn



Lets have a quick look at using Principal component analysis (PCA) an the Iris dataset.

What is PCA?

The simpilistic way to describe PCA is – it that it is one of many dimensionality reduction techniques, but its a very popular one. In fact, many of the tutorials or guides you find for machine learning typically use this technique in their work, even if its just for testing.

In short, if takes a lot of dimensions (variables) and reduces them to fewer. The key difference is that once the dataset it transformed the new variables become 'meaningless' or 'nameless'.

The two key places to use PCA (or any dimenstionality reduction technique) is too…

- Reduce the number of features you have – if the dataset is too broad and you perhaps want to train a ML model quicker.
- Visualisation – we can only really visualise data in 3 dimensions, so PCA can be good to reduce higher dimensions to 2 or 3. Typically most people just display as 2D.

A more detailed explanation of PCA can be found on Page 65 – [Learning scikit-learn: Machine Learning in Python].

Plan

Our plan…

- Load the IRIS dataset (4 features and 1 target)
- Visualise the dataset
- Build the PCA model

- Transform the IRIS data to 2 dimensions
- Visualise again

Load the data

The first step is to load the libraries you need. Here I am using the Anaconda distrubtion of Python 3, so it has everything I need already.

```
import numpy asnp
import pandas aspd
import matplotlib
import matplotlib.pyplot asplt
%matplotlibinline
from sklearn.decomposition import PCA
from sklearn import datasets



```

NOTE: %matplotlib inline – it is because I am doing the work in Jupyter Notebooks

I'll set some options too, to stop pandas displaying too much (or too little) data.

```
# Pandas Options (optional)
pd_maxrows=10
pd.set_option("display.max_rows",pd_maxrows)
pd.set_option("display.max_columns",30)

```

Next we actually load the data. The ***'from sklearn import datasets'*** contains the dataset so loading it is easy.

We also split it into *X* for the input variables and *y* for the classes.

```
iris=datasets.load_iris()
X=iris.data
y=iris.target
```

Lets have a look at the feature names and the class labels.

```
print(iris.feature_names)
print(iris.target_names)

```

You sould see…

['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

and

['setosa' 'versicolor' 'virginica']

Clean the Dataset

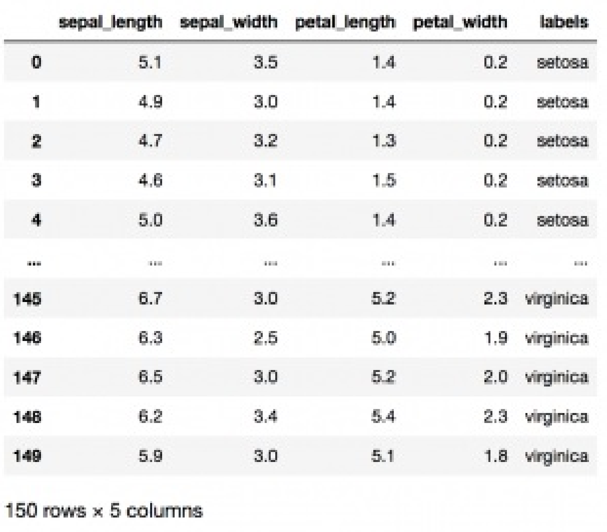
It doesn't need cleaning as such, but I like to work in Pandas Dataframes with small datasets. It lets you see what you are doing a little clearer.

Here we convert the data (*X*) to a dataframe and add the feature names (minus spaces and units of measure). Next we add the class labels.

```
df=pd.DataFrame(X)
df.columns=['sepal_length','sepal_width','petal_length','petal_width']
# Add the class names
df['labels']=iris.target_names[y]
df


```

You should see…



	sepal_length	sepal_width	petal_length	petal_width	labels
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

Visualisation

Now we are ready to plot the data.

First lets get the unique label names.

```
labels=df['labels'].unique().tolist()
```

This will give you…

['setosa', 'versicolor', 'virginica']

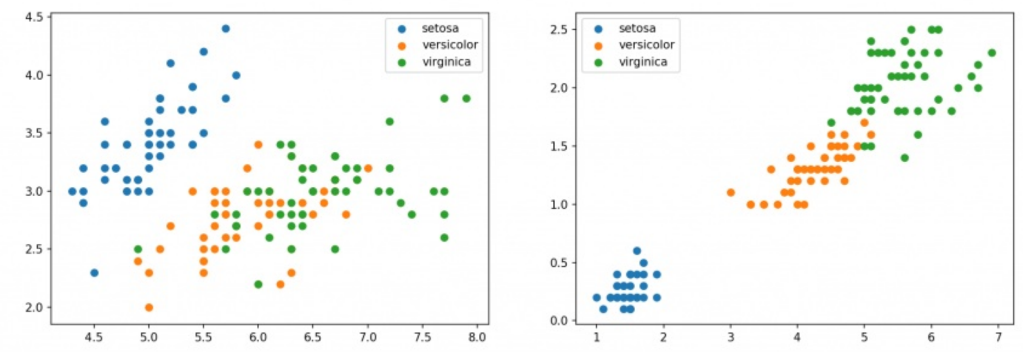
We will loop through these and plot each group (helps colour them up too).

```
plt.figure(figsize=(15,5))
plt.subplot(121)
forlab inlabels:
plt.scatter(df.loc[df['labels']==lab,'sepal_length'],df.loc[df['labels']==lab,'sepal_width'],label=lab)
plt.legend()
plt.subplot(122)
forlab inlabels:
plt.scatter(df.loc[df['labels']==lab,'petal_length'],df.loc[df['labels']==lab,'petal_width'],label=lab)
plt.legend()

```

Here we created 2 subplots, the first looking at sepal values and the second looking at petal values. Really we could actually look at each dimension against each other, a scatter matrix is good for that (See: Seaborn).

Out plot looks like this…



I'm not going to comment on much here, you can see how each iris is represented. Perhaps the axis labels would have been a nice addition.

PCA

Now lets do the PCA.

First I put the features back in there own array. I didn't really need to do this, but I like X to be the inputs.

```
X=df.iloc[:,0:4]
```

Next we create the PCA model which only really needs the number of componets, here we are converting the data (*X*) down to 2 features.

We then fit the PCA model and the use the model to transform the data. I save the transformed data as (*X_*).

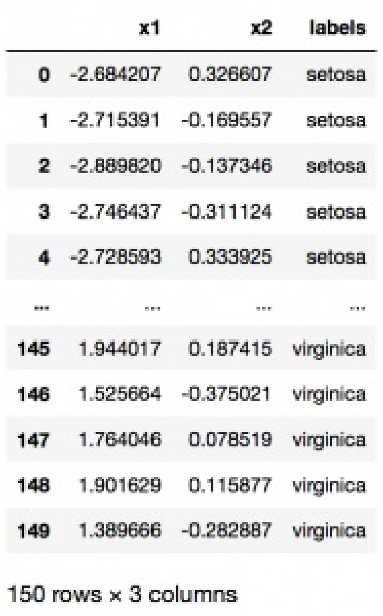
```
pca=PCA(n_components=2)
pca.fit(X)
X_=pca.transform(X)
```

Add it all back into a Dataframe, mine is called ***dfPCA***.

```
dfPCA=pd.DataFrame({'x1':X_[:,0],'x2':X_[:,1]})
dfPCA['labels']=df['labels']
dfPCA
```

We now have this…

were we can see that we only have 2 features (*x1* and *x2*) and the class label.



	x1	x2	labels
0	-2.684207	0.326607	setosa
1	-2.715391	-0.169557	setosa
2	-2.889820	-0.137346	setosa
3	-2.746437	-0.311124	setosa
4	-2.728593	0.333925	setosa
...	...	...	...
145	1.944017	0.187415	virginica
146	1.525664	-0.375021	virginica
147	1.764046	0.078519	virginica
148	1.901629	0.115877	virginica
149	1.389666	-0.282887	virginica

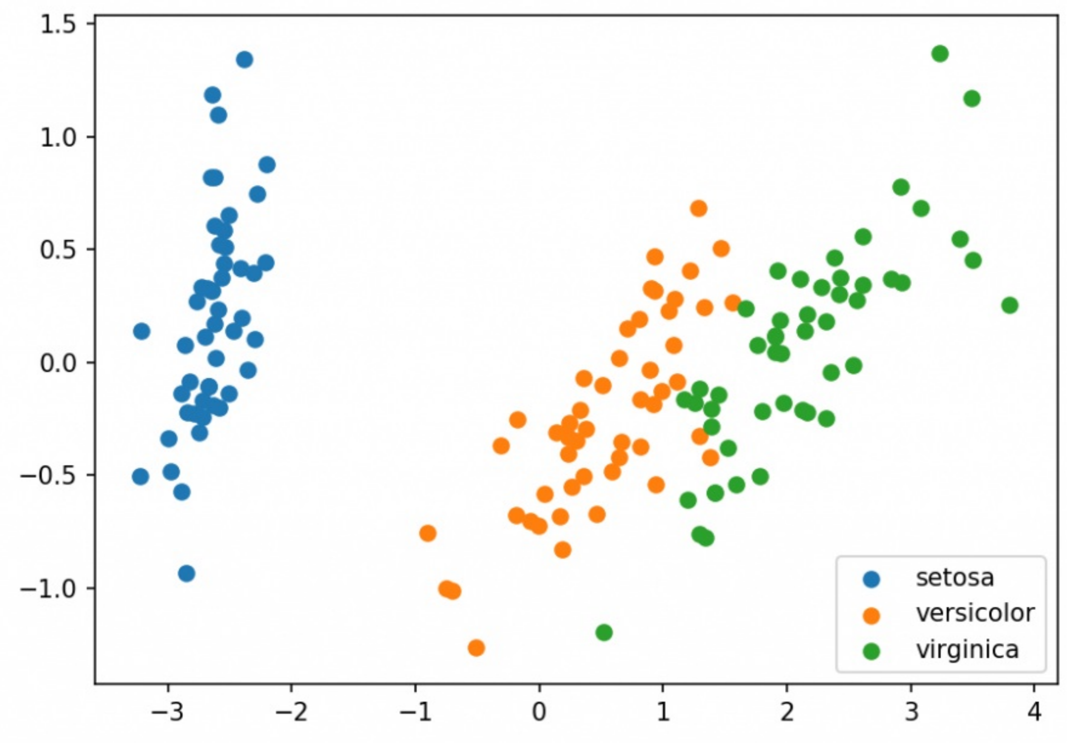
150 rows × 3 columns

Plot the PCA data

Finally lets plot the PCA features…

```
plt.figure(figsize=(7,5))
forlab inlabels:
plt.scatter(dfPCA.loc[dfPCA['labels']==lab,'x1'],dfPCA.loc[dfPCA['labels']==lab,'x2'],label=lab)
plt.legend()
```

which gives us…



Conclusion

The results for the PCA transformation has worked well, with this data. You should be able to use a range of different classifiers on this new data and they should perform well.