**Principle Component Analysis in Python**

Principle component analysis (PCA) is an unsupervised statistical technique that is used for dimensionality reduction.

It turns possible correlated features into a set of linearly uncorrelated ones called ‘Principle Components’.

In this post we’ll be doing PCA on the [pokemon data set](https://www.kaggle.com/abcsds/pokemon).

In [1]:

**import** **pandas** **as** **pd**

**from** **sklearn.decomposition** **import** PCA

pokemon = pd.read\_csv('data/pokemon.csv')

In [2]:

**print**(pokemon.head())

# Name Type 1 Type 2 Total HP Attack Defense \

0 1 Bulbasaur Grass Poison 318 45 49 49

1 2 Ivysaur Grass Poison 405 60 62 63

2 3 Venusaur Grass Poison 525 80 82 83

3 3 VenusaurMega Venusaur Grass Poison 625 80 100 123

4 4 Charmander Fire NaN 309 39 52 43

Sp. Atk Sp. Def Speed Generation Legendary

0 65 65 45 1 False

1 80 80 60 1 False

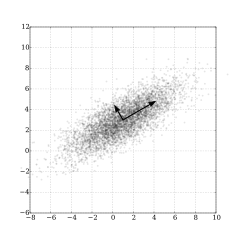
2 100 100 80 1 False

3 122 120 80 1 False

4 60 50 65 1 False

PCA is a good starting point for complex data. It models a linear subspace of the data by capturing the greatest variability. It does this by assessing the data’s covariance structure using matrix calculations and eigenvectors to compute the best unique features to describe the samples.

The first step finds the mean of the data, then search for the direction with the most variance. This direction is the principle component vectors, so it is added to a list. The next principle component is the orthogonal direction that has the next highest variance and so on.



This has a lot of practical uses including reducing the number of features you are working with for more processor intensive applications and noise reduction.

PCA is sensitive to the scale of features but, luckily for us on this occasion, our features are all of similar scale.

In [3]:

# Just take these features of interest

df = pokemon[['HP', 'Attack', 'Defense', 'Sp. Atk', 'Sp. Def', 'Speed']]

df.describe()

Out[3]:

|  | **HP** | **Attack** | **Defense** | **Sp. Atk** | **Sp. Def** | **Speed** |
| --- | --- | --- | --- | --- | --- | --- |
| **count** | **800.000000** | **800.000000** | **800.000000** | **800.000000** | **800.000000** | **800.000000** |
| **mean** | **69.258750** | **79.001250** | **73.842500** | **72.820000** | **71.902500** | **68.277500** |
| **std** | **25.534669** | **32.457366** | **31.183501** | **32.722294** | **27.828916** | **29.060474** |
| **min** | **1.000000** | **5.000000** | **5.000000** | **10.000000** | **20.000000** | **5.000000** |
| **25%** | **50.000000** | **55.000000** | **50.000000** | **49.750000** | **50.000000** | **45.000000** |
| **50%** | **65.000000** | **75.000000** | **70.000000** | **65.000000** | **70.000000** | **65.000000** |
| **75%** | **80.000000** | **100.000000** | **90.000000** | **95.000000** | **90.000000** | **90.000000** |
| **max** | **255.000000** | **190.000000** | **230.000000** | **194.000000** | **230.000000** | **180.000000** |

We will be reducing the features above down to just 2 principle components.

In [4]:

**from** **sklearn.decomposition** **import** PCA

pca = PCA(n\_components=2, svd\_solver='full')

pca.fit(df)

Out[4]:

PCA(copy=True, iterated\_power='auto', n\_components=2, random\_state=None,

svd\_solver='full', tol=0.0, whiten=False)

In [5]:

T = pca.transform(df)

In [6]:

# Started with 6 dimensions

df.shape

Out[6]:

(800, 6)

In [7]:

# Left with 2 principle components

T.shape

Out[7]:

(800, 2)

In [8]:

df.head()

Out[8]:

|  | **HP** | **Attack** | **Defense** | **Sp. Atk** | **Sp. Def** | **Speed** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | **45** | **49** | **49** | **65** | **65** | **45** |
| **1** | **60** | **62** | **63** | **80** | **80** | **60** |
| **2** | **80** | **82** | **83** | **100** | **100** | **80** |
| **3** | **80** | **100** | **123** | **122** | **120** | **80** |
| **4** | **39** | **52** | **43** | **60** | **50** | **65** |

In [9]:

T

Out[9]:

array([[ -45.86072754, -5.38443151],

[ -11.15293667, -5.80561951],

[ 36.94600862, -5.23612965],

...,

[ 75.99988475, -27.27078641],

[ 114.0967126 , -36.87056714],

[ 72.88355049, 15.15261625]])

We can use the explained\_variance\_ratio\_ method of our principle component analysis object to see how much of the variance is explained by each of our principle components vectors.

In [10]:

pca.explained\_variance\_ratio\_

Out[10]:

array([ 0.46096131, 0.18752145])

So just two principle components can explain almost 65% of the variance from these 6 features.

**Interpreting Components**

We can access the correlations between the components and original variables using the components\_ method of our PCA() object.

Interpretation of these relies on finding the most highly correlated components (for this example we’ll use a cut-off of 0.45)

In [11]:

components = pd.DataFrame(pca.components\_, columns = df.columns, index=[1, 2])

components

Out[11]:

|  | **HP** | **Attack** | **Defense** | **Sp. Atk** | **Sp. Def** | **Speed** |
| --- | --- | --- | --- | --- | --- | --- |
| **1** | **0.300808** | **0.492892** | **0.380635** | **0.508981** | **0.394370** | **0.327263** |
| **2** | **0.042210** | **0.076545** | **0.695216** | **-0.383311** | **0.173894** | **-0.576079** |

So for the first principle component, Attack and Sp. Atk is significant so this principle component is correlated well with Attack and Sp. Atk and pokemon with a high value for the first principle component have high Attack and Sp. Atk.

For the second principle component, this will increase with an increase in Defense and a decrease in Speed. Pokemon with high values of the second principle component will have a high value for Defense but a low value for speed.

We can do some mathematics to find out which are the most important features:

In [12]:

**import** **math**

**def** get\_important\_features(transformed\_features, components\_, columns):

"""

This function will return the most "important"

features so we can determine which have the most

effect on multi-dimensional scaling

"""

num\_columns = len(columns)

# Scale the principal components by the max value in

# the transformed set belonging to that component

xvector = components\_[0] \* max(transformed\_features[:,0])

yvector = components\_[1] \* max(transformed\_features[:,1])

# Sort each column by it's length. These are your \*original\*

# columns, not the principal components.

important\_features = { columns[i] : math.sqrt(xvector[i]\*\*2 + yvector[i]\*\*2) **for** i **in** range(num\_columns) }

important\_features = sorted(zip(important\_features.values(), important\_features.keys()), reverse=True)

**print** "Features by importance:**\n**", important\_features

get\_important\_features(T, pca.components\_, df.columns.values)

Features by importance:

[(143.62419952151768, 'Defense'), (119.74350606922016, 'Speed'), (105.83113958361301, 'Sp. Atk'), (76.02281561178808, 'Attack'), (68.1790434253425, 'Sp. Def'), (46.24128335926672, 'HP')]

We see that the most significant features for this PCA are Defence, Speed, Special Attack and Attack, as we saw when examining the components\_ previously.

By plotting these lengths, we can see this visually:

In [13]:

**import** **matplotlib.pyplot** **as** **plt**

%**matplotlib** inline

plt.style.use('ggplot')

**def** draw\_vectors(transformed\_features, components\_, columns):

"""

This funtion will project your \*original\* features

onto your principal component feature-space, so that you can

visualize how "important" each one was in the

multi-dimensional scaling

"""

num\_columns = len(columns)

# Scale the principal components by the max value in

# the transformed set belonging to that component

xvector = components\_[0] \* max(transformed\_features[:,0])

yvector = components\_[1] \* max(transformed\_features[:,1])

ax = plt.axes()

**for** i **in** range(num\_columns):

# Use an arrow to project each original feature as a

# labeled vector on your principal component axes

plt.arrow(0, 0, xvector[i], yvector[i], color='b', width=0.0005, head\_width=0.02, alpha=0.75)

plt.text(xvector[i]\*1.2, yvector[i]\*1.2, list(columns)[i], color='b', alpha=0.75)

**return** ax

In [14]:

ax = draw\_vectors(T, pca.components\_, df.columns.values)

T\_df = pd.DataFrame(T)

T\_df.columns = ['component1', 'component2']

T\_df['color'] = 'y'

T\_df.loc[T\_df['component1'] > 125, 'color'] = 'g'

T\_df.loc[T\_df['component2'] > 125, 'color'] = 'r'

plt.xlabel('Principle Component 1')

plt.ylabel('Principle Component 2')

plt.scatter(T\_df['component1'], T\_df['component2'], color=T\_df['color'], alpha=0.5)

plt.show()

We can see from the plot that all components are positive in the first principle component but speed and special attack in the second principle component are negative. Their lengths portray their magnitudes.

The pokemon in green have high values for the first principle component – They have high Attack and Sp. Atk

The pokemon in red have high values for the second principle component – They have high Defense and low Speed

In [15]:

# High Attack, High Sp. Atk, all of these pokemon are legendary

**print**(pokemon.loc[T\_df[T\_df['color'] == 'g'].index])

# Name Type 1 Type 2 Total HP Attack \

163 150 MewtwoMega Mewtwo X Psychic Fighting 780 106 190

164 150 MewtwoMega Mewtwo Y Psychic NaN 780 106 150

422 382 KyogrePrimal Kyogre Water NaN 770 100 150

424 383 GroudonPrimal Groudon Ground Fire 770 100 180

426 384 RayquazaMega Rayquaza Dragon Flying 780 105 180

Defense Sp. Atk Sp. Def Speed Generation Legendary

163 100 154 100 130 1 True

164 70 194 120 140 1 True

422 90 180 160 90 3 True

424 160 150 90 90 3 True

426 100 180 100 115 3 True

In [16]:

# High Defense, Low Speed

**print**(pokemon.loc[T\_df[T\_df['color'] == 'r'].index])

# Name Type 1 Type 2 Total HP Attack Defense \

224 208 SteelixMega Steelix Steel Ground 610 75 125 230

230 213 Shuckle Bug Rock 505 20 10 230

333 306 AggronMega Aggron Steel NaN 630 70 140 230

Sp. Atk Sp. Def Speed Generation Legendary

224 55 95 30 2 False

230 10 230 5 2 False

333 60 80 50 3 False