14) PCA

- Stands for Principal Component Analysis
- It is very sensitive to scaling, so we should scale our dataset prior to applying it
- MUST SCALE DATA

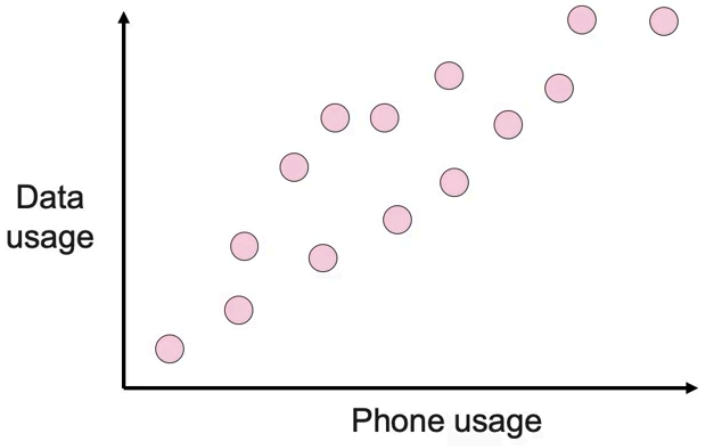
How it works

Suppose we have the following dataset. As we can see, the two features are very correlated

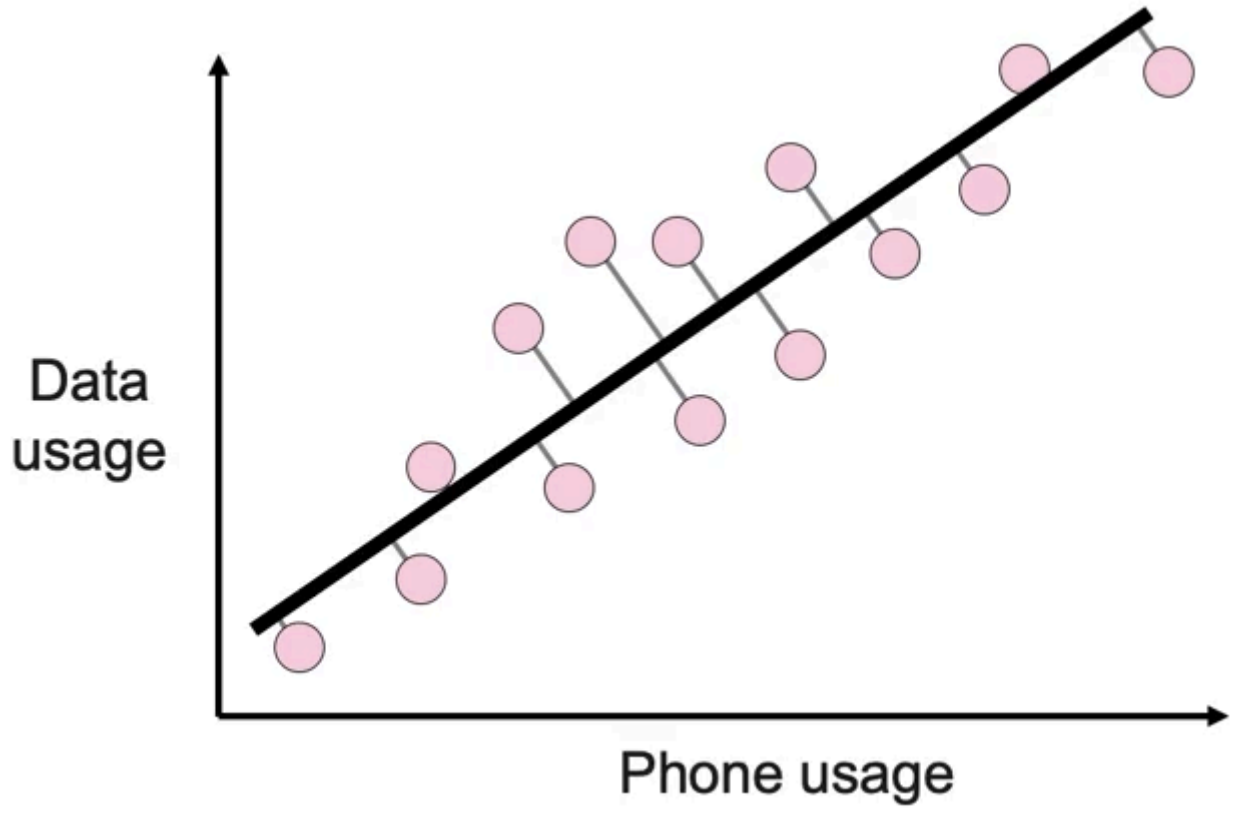
– Two features:
Phone usage (minutes) and data usage.

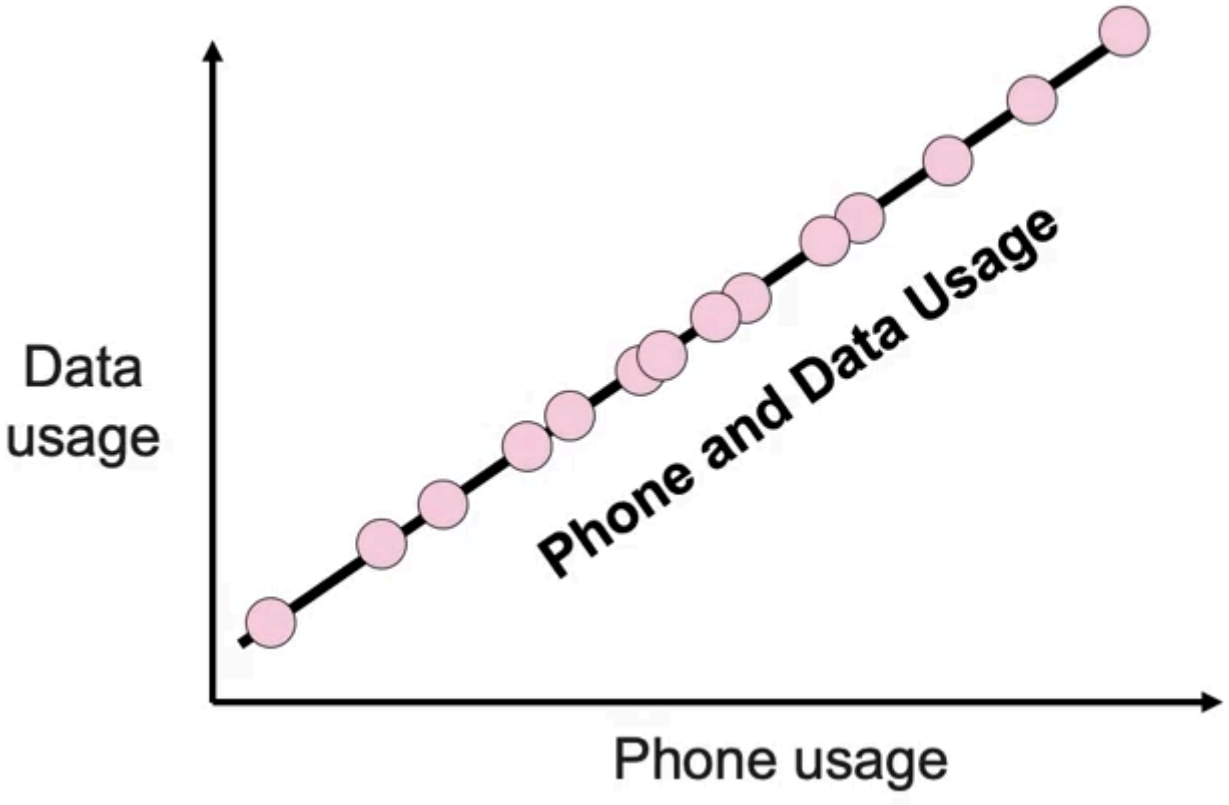
– Both features increase together (correlated).

– Can we reduce number of features to one?

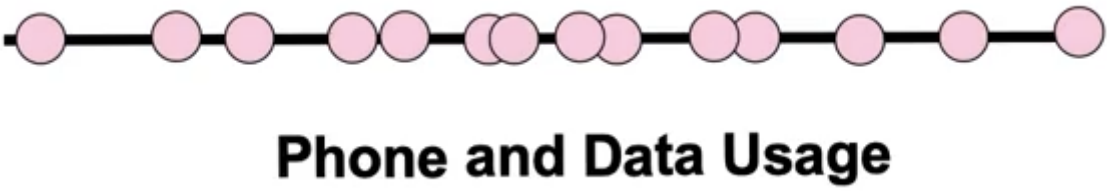


What we can do is we can consider this line and project the points onto it, like so:





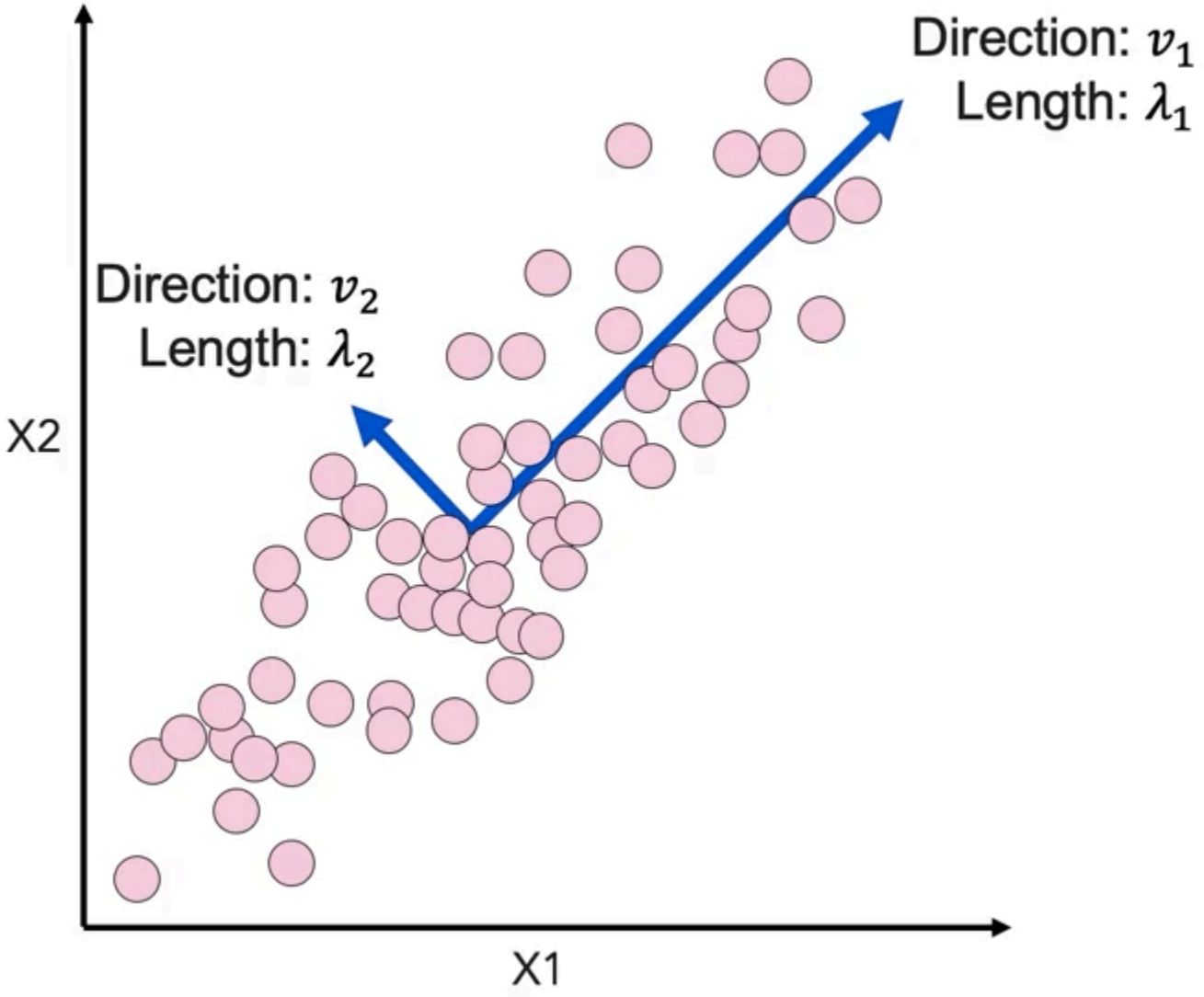
We can think of his transformation as "the scaled addition" of both the columns. So we now have only one column, which was created as the combination of the two original columns



Phone and Data Usage

Maths behind this

First we gotta understand what a "right singular vector" is. These are basically the lines on which we will project our data. Using linear algebra, we can find the "primary right singular vector" and the "secondary right singular vector". They both are perpendicular to other. So they might like look so:

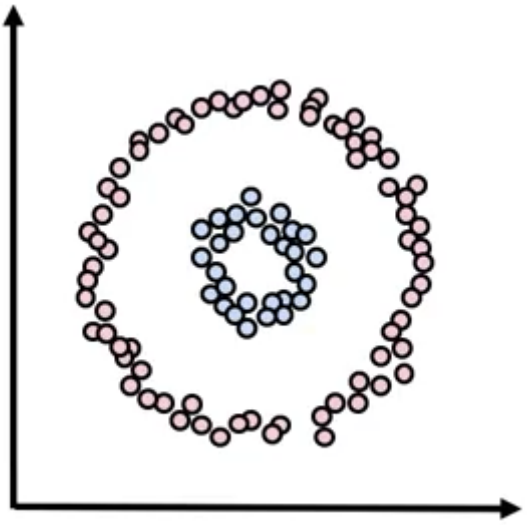


We use SVD (single value decomposition) to find these vectors

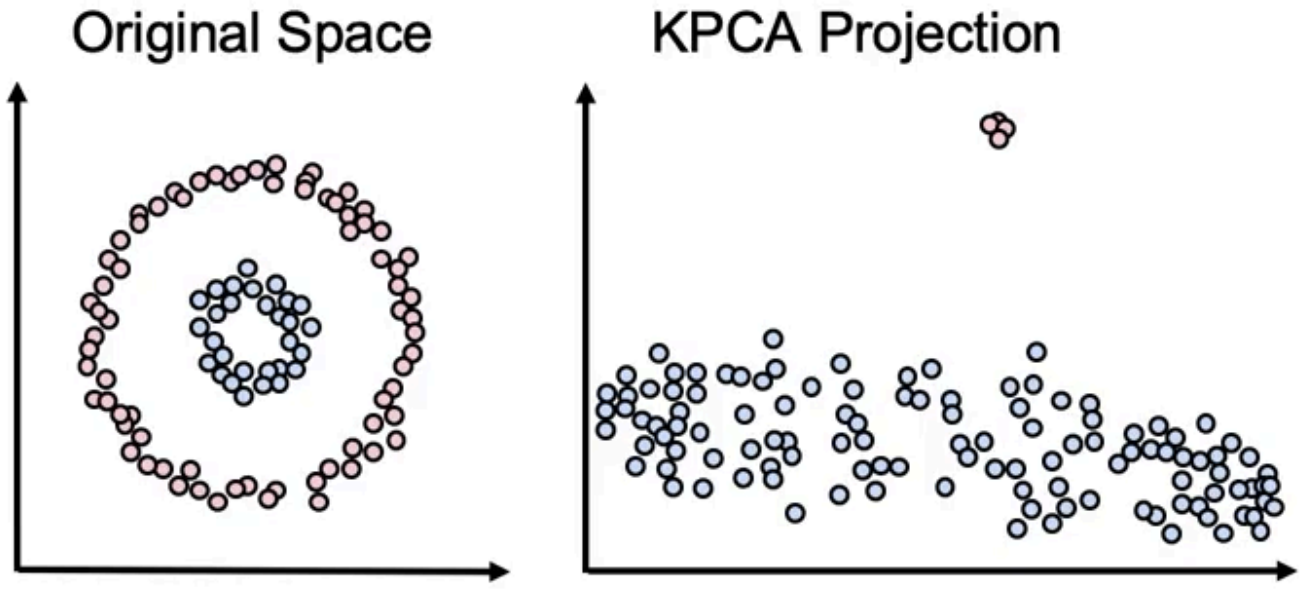
Nonlinear PCA (ie KPCA)

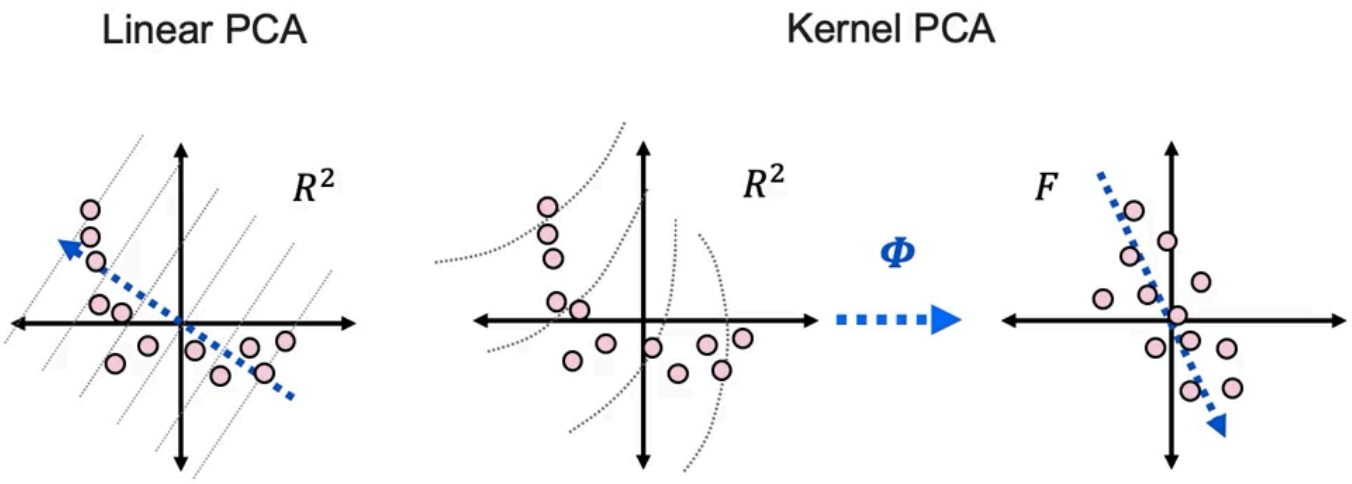
- Stands for "Kernel Principal Component Analysis"
- Here we basically apply a kernel (like the ones in SVMs) to "flatten" out the data

For example, normal PCA cannot be used on this dataset:



So we use KPCA projection to come up with a "linearly separable"





Code

```
from sklearn.decomposition import PCA, KernelPCA

model = PCA(n_components=3) # n_components in the number of features end data should have
X_trans = model.fit_transform(X_orig)

model = KernelPCA(n_components=3, kernel='rbf', gamma=1.0)
X_trans = model.fit_transform(X_orig)
```