Principal component analysis

Input: a matrix X, unlabeled data matrix

Output: k directions of maximum variation

PCA is a data feature reduction technique that can be especially useful in machine learning applications. As a general data analysis tool, PCA yields directions of maximum variance of the inputted data, and the corresponding amount of variance captured by each of those directions. We choose some k, a hyperparameter, which determines the number of vectors or 'directions' PCA returns.

As a dimensionality reduction tool, often when we work with data in machine learning, the number of features (variables) can often be too numerous to work with easily. The general issue is of increased computational complexity. Instead, we can use PCA to choose the k most varied directions in the inputted data, and then project the input data matrix onto those directions, resulting in only k features.

Note on hyperparameters: hyperparameters, such as k in PCA, are not values determined by the algorithm to be the best for the problem. As such we must think of ways to effectively validate our choice for k and compare how different hyperparameter values effect our model. Here for example, we would k to capture a large portion of the variance.

In [0]:
```
import numpy as np
```

In [0]:
```
def scratch_PCA(X, k):
    demeaned_X= X - np.mean(X, axis=0)
    #we now have to caculate the covariance matrix
    cov_X= np.matmul(np.transpose(demeaned_X), demeaned_X)
    sigma,V= np.linalg.eig(cov_X)
    # you then select the k biggest eigenvalues and their corresponding eigenvec
tors
    return sigma[:k], V[:k]
```

In [0]:
```
def svd_PCA(X,k):
  U, S, V = np.linalg.svd(X - np.mean(X, axis=0))
  return S[:k]**2, V[:k]
```

Explanation

In PCA we choose the first k components of the SVD of a matrix. In a nutshell, SVD represents the breakdown of the input into a sum of rank 1 matrices. By ordering the variance (the sigma values) by high to low, we capture the largest amount of variance in the first k components. As we increase k to the rank of the input, we fully capture the data matrix.

Mathematically:

SVD is represented as U *S* V.T

- U - eigenvectors of X @ X.T
- V - eigenvectors of X.T @ X
- S - sigma squared = eigenvalues of both

PCA uses the first k values of S and the first k vectors of V

In [0]:
```
from sklearn.decomposition import PCA
```

In practice we would use modules such as sci-kit learn for running methods such as PCA.

Example extrapolated method below:

In [0]:
```
def sk_PCA(X, k):
  # initialize pca with num of components (k)
  pca = PCA(n_components=k)

  # run pca on our data set
  pca.fit(X)

  # get our first k components from our solutions
  variance = pca.explained_variance_[:k]
  vectors = pca.components_[:k]
  vectors = [np.asarray(vectors[i]) for i in range(len(vectors))]

  return variance, vectors
```