Lecture 40 - Principal Component Analysis (PCA)

Feature Selection & Feature Extraction

Note that feature selection and feature extraction are two different approaches.

1. **Feature Selection:** Choosing a subset of the original pool of features.
2. **Feature Extraction:** Creating useful features from data.
    - Dimensionality reduction techniques such as PCA are performing feature extraction.

In **feature selection**, we can have three main approaches:

1. **Wrappers:** a wrapper evaluates a specific model sequentially using different potential subsets of features to get the subset that best works in the end. They are highly costly and have a high chance of overfitting, but also a high chance of success, on the other hand.
    - Example: Forward Sequential Feature Selection, Backward Feature Elimination, etc.
2. **Filters:** for a much faster alternative, filters do not test any particular algorithm, but rank the original features according to their relationship with the problem (labels) and just select the top of them. Correlation and mutual information are the most widespread criteria. There are many easy to use tools, like the feature selection `scikit-learn` package.
    - Example: Correlation, chi-squared test, ANOVA, information gain, etc.
3. **Embedded:** this group is made up of all the Machine Learning techniques that include feature selection during their training stage.
    - Example: LASSO, Elastic Net, Ridge, etc.

Principal Component Analysis (PCA)

A very common approach (and one of the simplest approaches) to **dimensionality reduction** is **Principal Component Analysis** (or **PCA**).

- PCA takes data from *sensor coordinates* to *data centric coordinates* using linear transformations.

PCA uses a **linear transformation** to **minimize the redundancy** of the resulting transformed data (by ending up with data that is uncorrelated).

- This means that every transformed dimension is more informative.
- In this approach, the dimensionality of the space is still the same as the original data, but the space of features are now arranged such that they contain the most information.

If we wish to reduce dimensionality of our feature space, we can choose only the features that carry over the most information in the linearly transformed space.

- In other words, PCA will find the underlying **linear manifold** that the data is embedded in.

PCA finds the directions of maximum variance in high-dimensional data and projects it onto a new subspace with equal or fewer dimensions than the original one.

There are a couple of points-of-view on how to find A:

1. Maximum Variance Formulation
2. Minimum-error Formulation

1. PCA as Maximum Variance Formulation

Consider the data X comprised on N data samples in a D-dimensional space, so X is a matrix of size $D \times N$.

The **first step** in PCA is to centralize or demean X. This will is to guarantee that all features will have the same impact and not weight more only because their range of values is much larger (example, age vs income).

- Without loss of generality, let's assume that we subtracted the mean to the input data, X. Now, X has zero mean.

The **second step** is to find the linear transformation A that transforms X to a space where features are:

1. Uncorrelated (preserve all dimensions)
2. reduced (dimensionality reduction)

$$Y = AX$$

where A is a $D \times D$ matrix, X is a $D \times N$ data matrix and therefore Y is also a $D \times N$ transformed data matrix.

The variance of the transformed data Y is given by:

$$\begin{aligned} R_y &= E[YY^T] \\ &= E[AX(AX)^T] \\ &= E[AXX^TA^T] \\ &= AE[XX^T]A^T \\ &= AR_xA^T \end{aligned}$$

Note that we are computing the variance along the dimensions of Y, therefore, R_y is a $D \times D$ matrix. Similarly, R_X represents the covariance of the data X. Covariances matrices are symmetric therefore $R_X = R_X^T$ and $R_Y = R_Y^T$.

Similarly, R_X represents the covariance of the data X.

If we write A in terms of vector elements:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} \overrightarrow{a_1} \\ \overrightarrow{a_2} \end{bmatrix}$$

Then,

$$\begin{aligned} R_y &= \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} R_X \begin{bmatrix} a_1 & a_2 \end{bmatrix} \\ &= \begin{bmatrix} a_1 R_X a_1^T & a_1 R_X a_2^T \\ a_2 R_X a_1^T & a_2 R_X a_2^T \end{bmatrix} \end{aligned}$$

- If we want to represent the data in a space in which the features are **uncorrelated**, what shape does the covariance matrix have to take?

Diagonal! Why?

- Can we use the eigenvectors of R_X as our linear transformation A?

Consider the case where we are trying to project the data X into a 1-dimensional space, so we are trying to find the direction a_1 where maximal data variance is preserved:

$$\arg_{a_1} \max a_1 R_X a_1^T$$

We want this solution to be bounded (considering $a_1 = \infty$ would maximize), so we need to constraint the vector to have norm 1

$$\|a_1\|_2^2 = 1 \iff a_1 a_1^T = 1$$

Then, we using Lagrange Optimization:

$$\mathcal{L} = a_1 R_X a_1^T + \lambda_1 (1 - a_1 a_1^T)$$

Solving for a_1:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial a_1} &= 0 \\ R_X a_1^T + R_X^T a_1^T - 2\lambda_1 a_1^T &= 0 \\ R_X a_1^T &= \lambda a_1^T \end{aligned}$$

- Does this look familiar?

This is stating that a_1^T must be an eigenvector of R_X!

So coming back to the question "Can we use the eigenvectors of X as our linear transformation A?" YES!

- If we left multiply by a_1 and make use of $a_1 a_1^T = 1$:

$$a_1 R_X a_1^T = \lambda_1$$

So the variance will be maximum when we set the project direction a_1 equal to the eigenvector having the largest eigenvalue λ_1.

- This eigenvector is known as the firt **principal component**.
- As you may anticipate, the linear trasnformation A will the be a matrix whose row entries are **sorted** the eigenvectors (sorted by their correspondent eigenvalue).

$$A = \begin{bmatrix} \overrightarrow{a_1} \\ \overrightarrow{a_2} \end{bmatrix}$$

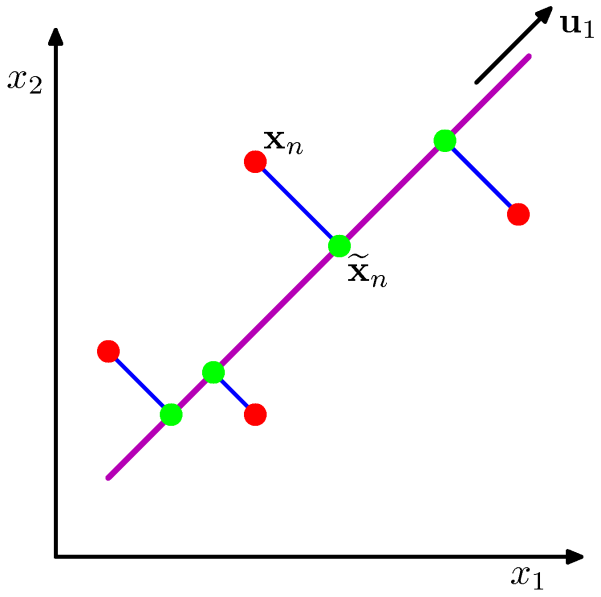
where a_1 and a_2 are eigenvectors of R_X with correspondent eigenvalues λ_1 and λ_2 with $\lambda_1 > \lambda_2$.

2. PCA as Minimum-error Formulation

We can also look at PCA as a minimization of mean squared error.

In [1]:
```
from IPython.display import Image
Image('figures/Figure12.2.png',width=300)
```

Out[1]:



Consider X and a D-dimensional orthogonal basis $\mathbf{u}$:

$$\hat{x} = \sum_{i=1}^{m} y_i a_i$$

where $m < D$.

$$y_j = x^T a_j$$

where $A^T A = I$.

We want to minimize the residual error:

$$\epsilon = x - \hat{x} = \sum_{i=m+1}^{D} y_i a_i$$

- The objective function we will use is the mean square residual:

$$\begin{aligned} J &= E\left[\|\epsilon\|_2^2\right] \\ &= E\left[\left(\sum_{i=m+1}^{D} y_i a_i\right)\left(\sum_{i=m+1}^{D} y_i a_i\right)\right] \\ &= \sum_{j=m+1}^{D} E[y_j^2] \\ &= \sum_{j=m+1}^{D} E[(a_j^T \mathbf{x})(\mathbf{x}^T a_j)] \\ &= \sum_{j=m+1}^{D} a_j^T E[\mathbf{x}\mathbf{x}^T] a_j \\ &= \sum_{j=m+1}^{D} a_j^T R_x a_j \end{aligned}$$

Minimize the error and incorporate Lagrange parameters for $U^T U = I$:

$$\begin{aligned} \frac{\partial J}{\partial a_j} &= 2(R_x a_j - \lambda_j a_j) = 0 \\ R_x a_j &= \lambda_j a_j \end{aligned}$$

So, the sum of the error is the sum of the eigenvalues of the unused eigenvectors. So, we want to select the eigenvectors with the m largest eigenvalues.

Steps of PCA

Consider the data X:

1. Subtract the mean $\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$
2. Compute the covariance matrix R_X (by definition, the covariance already subtracts the data's mean)
3. Compute eigenvectors and eigenvalues of the matrix R_X, and store the sorted eigenvectors (e_i) in decreasing eigenvalue (λ_i) order.
4. The linear transformation A will be: $A = \begin{bmatrix} \mathbf{a_1} \\ \mathbf{a_2} \end{bmatrix}$, where $\lambda_1 > \lambda_2$
5. Projection: $y = Ax$

Note that the formal definition of covariance already accounts for demeaning the data.

In [1]:

```
import numpy as np
import numpy.random as npr
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('bmh')
```

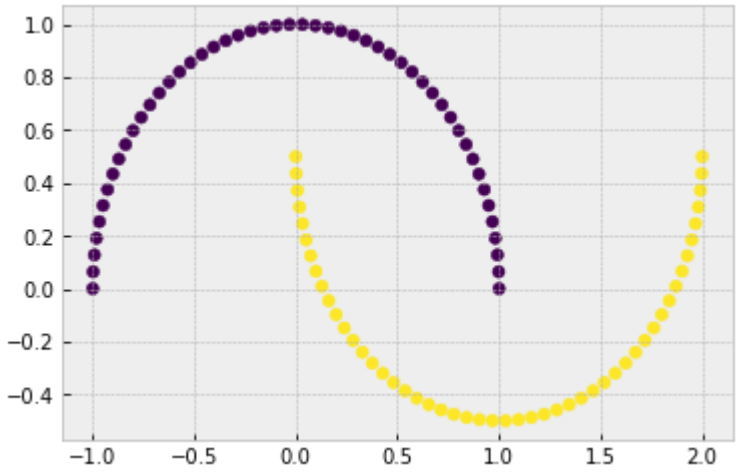
In [2]:

```
def PCA(X, m, display=1):
    '''This function implements PCA. The data matrix X is DxN matrix,
    where D is the dimension and N the number of points'''

    D, N = X.shape

    # Demean the Data
    data = X - X.mean(axis=1).reshape(-1, 1)

    # Covariance of the input data X
    cov_mat = np.cov(data)

    # Find eigenvectors and eigenvalues
    eigen_vals, eigen_vecs = np.linalg.eig(cov_mat)

    # Sort eigenvectors by magnitude of eigenvalues
    eigen_pairs = [(np.abs(eigen_vals[i]), eigen_vecs[:,i]) for i in range(len(eigen_va
    eigen_pairs = sorted(eigen_pairs, key=lambda x: x[0], reverse=True)

    # Linear transformation
    A = np.hstack(([eigen_pairs[i][1][:, np.newaxis] for i in range(m)]))

    #compute explained variance and plot
    cum_var_exp=0
    if display:
        total = sum(eigen_vals)
        var_explained = [(i/total) for i in sorted(eigen_vals, reverse=True)]
        cum_var_exp = np.cumsum(var_explained)
        plt.bar(range(1,D+1), var_explained, alpha=0.5, align='center', label='individu
        plt.step(range(1,D+1), cum_var_exp, alpha=0.5, where='mid', label='cumulative e
        plt.ylabel('Explained variance ratio')
        plt.xlabel('Principal components')
        plt.legend(loc='best')
        plt.show();
    return A, eigen_pairs, cum_var_exp
```

Example: Half-Moons

In [3]:

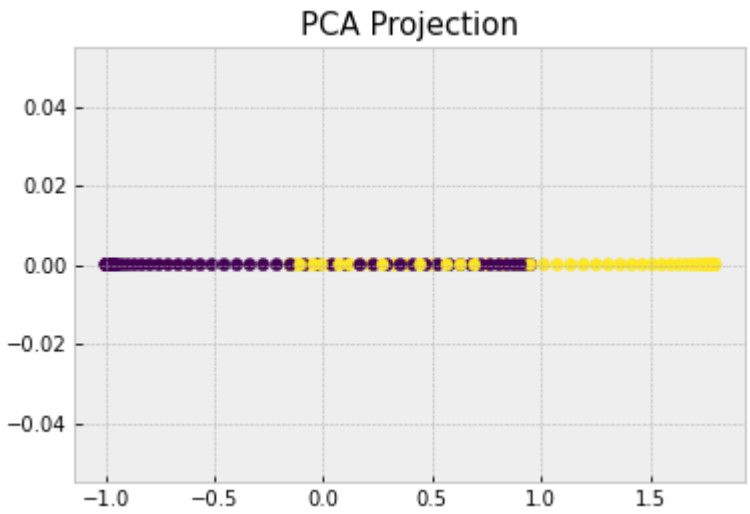
```
from sklearn.datasets import make_moons

X, y = make_moons(n_samples = 100, random_state = 123)

plt.scatter(X[:,0], X[:,1], c=y);
```



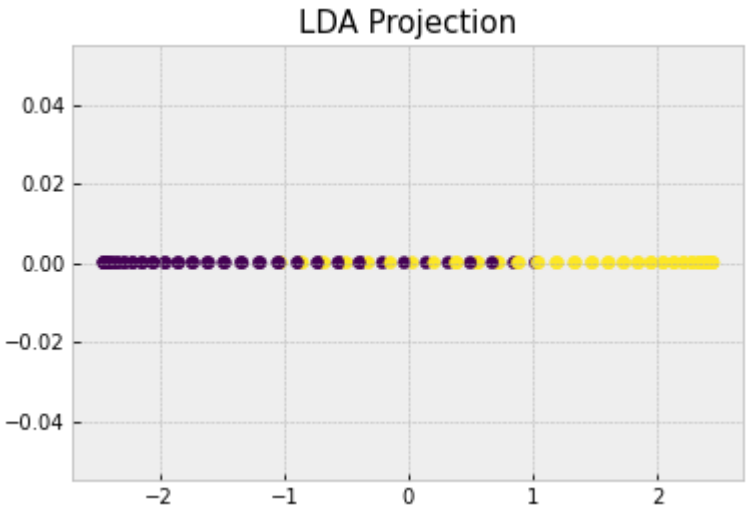
In [5]:
```
X.shape
```

Out[5]:
```
(100, 2)
```

In [6]:
```
A,_,_ = PCA(X.T,1,0)

projection_PCA = X@A #or (A.T@X.T).T

plt.scatter(projection_PCA, np.zeros(len(projection_PCA)), c=y)
plt.title('PCA Projection', fontsize=15);
```



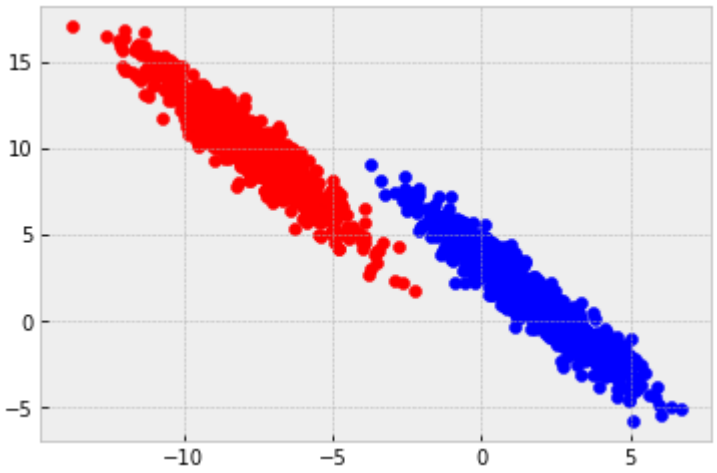
In [7]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

model = LDA(n_components=1)
projection_LDA = model.fit_transform(X,y)

plt.scatter(projection_LDA, np.zeros(len(projection_LDA)), c=y)
plt.title('LDA Projection', fontsize=15);
```



Example: Synthetic oval-shaped data

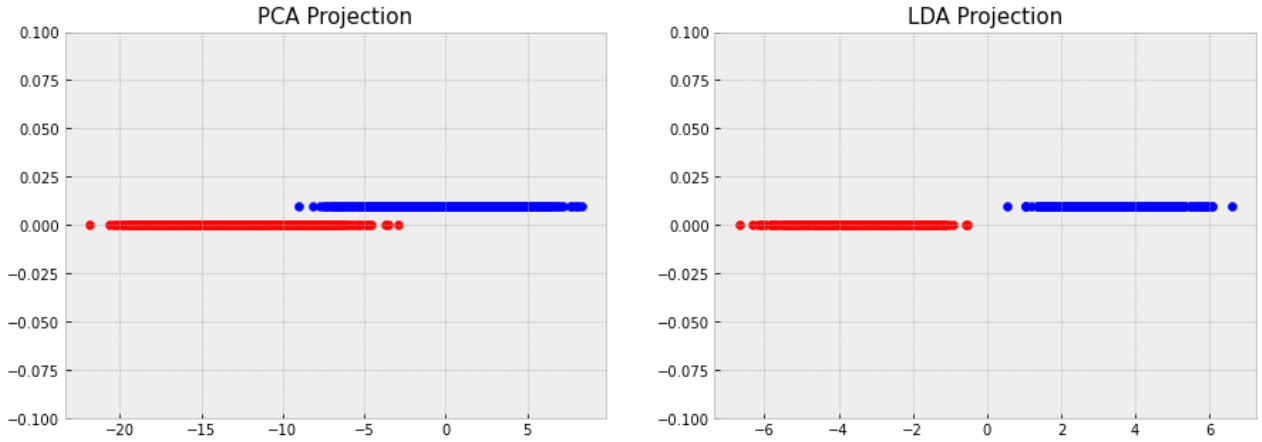
In [8]:

```
from sklearn import datasets
X, y = datasets.make_blobs(n_samples=1500,centers=2,cluster_std=2.5)
X    = np.dot(X, [[0.60834549, -0.63667341], [-0.40887718, 0.85253229]])
plt.scatter(X[y==0,0],X[y==0,1],c='r')
plt.scatter(X[y==1,0],X[y==1,1],c='b');
```



In [9]:

```
A,_,_ = PCA(X.T,1,0)
projection_PCA = X@A

model = LDA(n_components=1)
projection_LDA = model.fit_transform(X,y)

plt.figure(figsize=(15,5))
plt.subplot(121)
plt.scatter(projection_PCA[y==0], np.zeros(len(projection_PCA[y==0])), c='r')
plt.scatter(projection_PCA[y==1], 0.01+np.zeros(len(projection_PCA[y==1])), c='b')
plt.ylim([-0.1,0.1]);plt.title('PCA Projection', fontsize=15);
plt.subplot(122)
plt.scatter(projection_LDA[y==0], np.zeros(len(projection_LDA[y==0])), c='r')
plt.scatter(projection_LDA[y==1], 0.01+np.zeros(len(projection_LDA[y==1])), c='b')
plt.ylim([-0.1,0.1]); plt.title('LDA Projection', fontsize=15);
```



Observation: LDA finds the direction of projection that maximize class separability.

Example: Wine Dataset

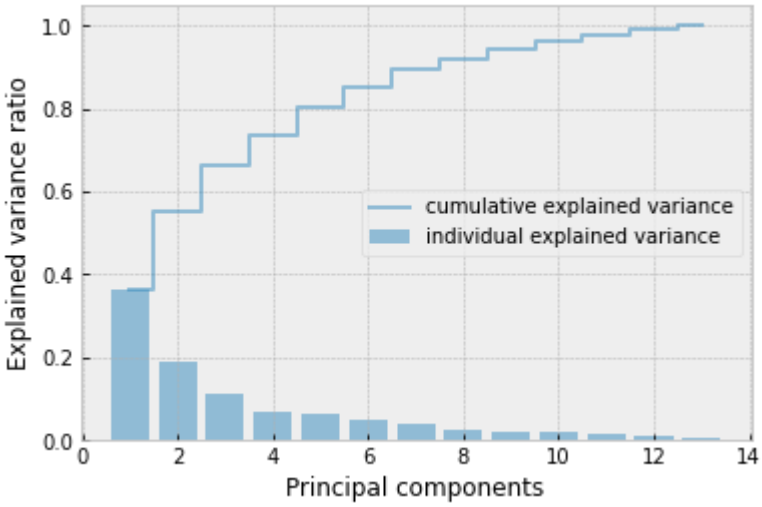
https://archive.ics.uci.edu/ml/datasets/wine

In [10]:
```
from sklearn.datasets import load_wine

wine_data = load_wine(return_X_y=False)
# print(wine_data.DESCR)
```

In [11]:
```
from sklearn.preprocessing import StandardScaler

X = wine_data.data
y = wine_data.target

sc = StandardScaler()
X = sc.fit_transform(X)

print(X.shape, y.shape)
```

```
(178, 13) (178,)
```

In [12]:
```
A, _, cum_var_exp= PCA(X.T,2,1)

projection = X@A
```



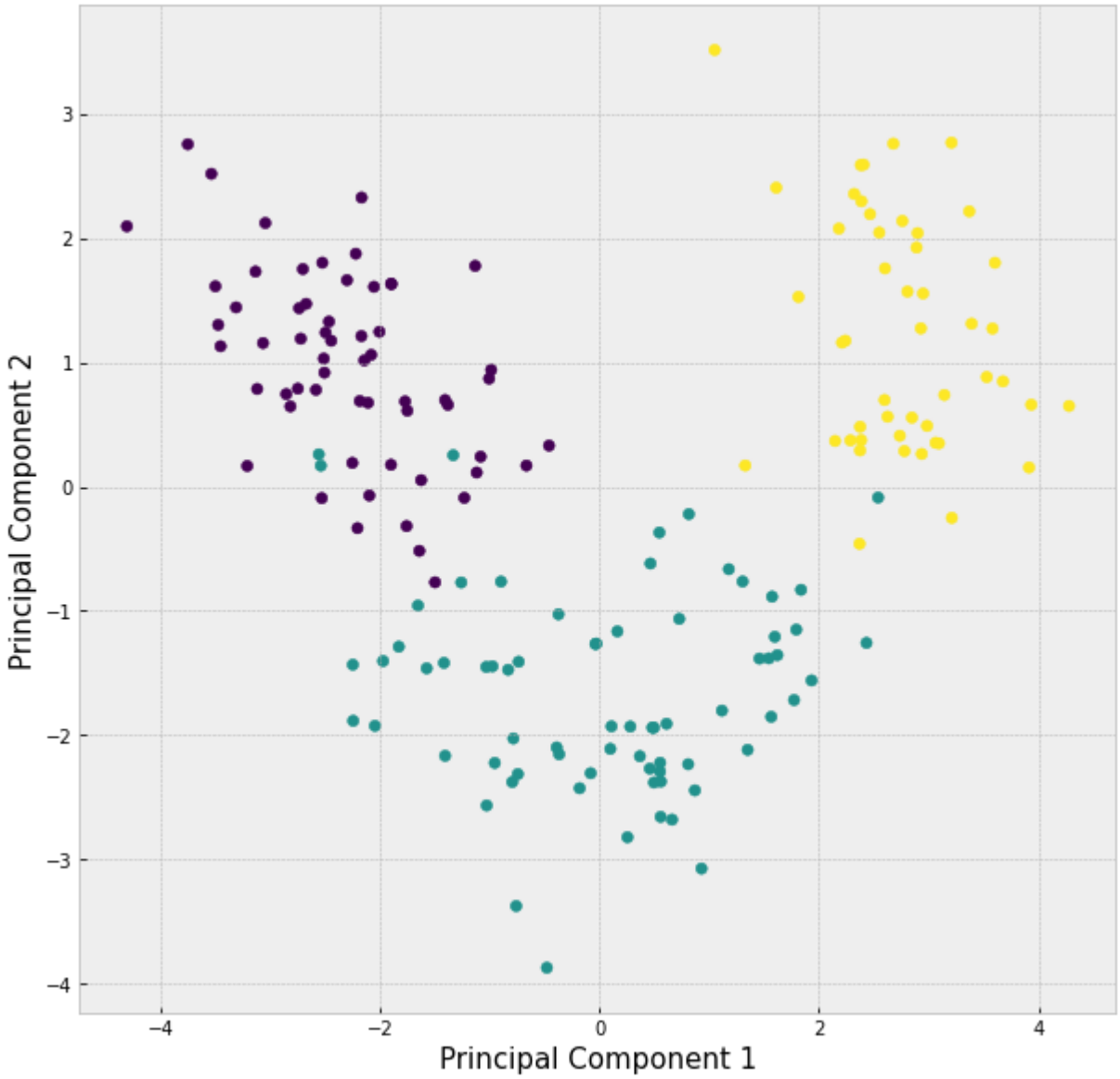
In [13]:
```
cum_var_exp
```

Out[13]:
```
array([0.36198848, 0.55406338, 0.66529969, 0.73598999, 0.80162293,
       0.85098116, 0.89336795, 0.92017544, 0.94239698, 0.96169717,
       0.97906553, 0.99204785, 1.        ])
```

In [14]:
```
# Finds the dimensionality that preserves at least 90% of the explained variance

np.where(cum_var_exp>=0.9)[0]

# smallest number is 7
```

Out[14]:
```
array([ 7,  8,  9, 10, 11, 12], dtype=int64)
```

In [15]:
```
# Visualization of projection to first 2 dimensions.

fig, ax = plt.subplots(figsize=(10, 10))
plt.scatter(projection[:, 0], projection[:, 1], c=y)
plt.xlabel('Principal Component 1', fontsize=15)
plt.ylabel('Principal Component 2', fontsize=15);
```



Example: MNIST Dataset Handwritten Digits

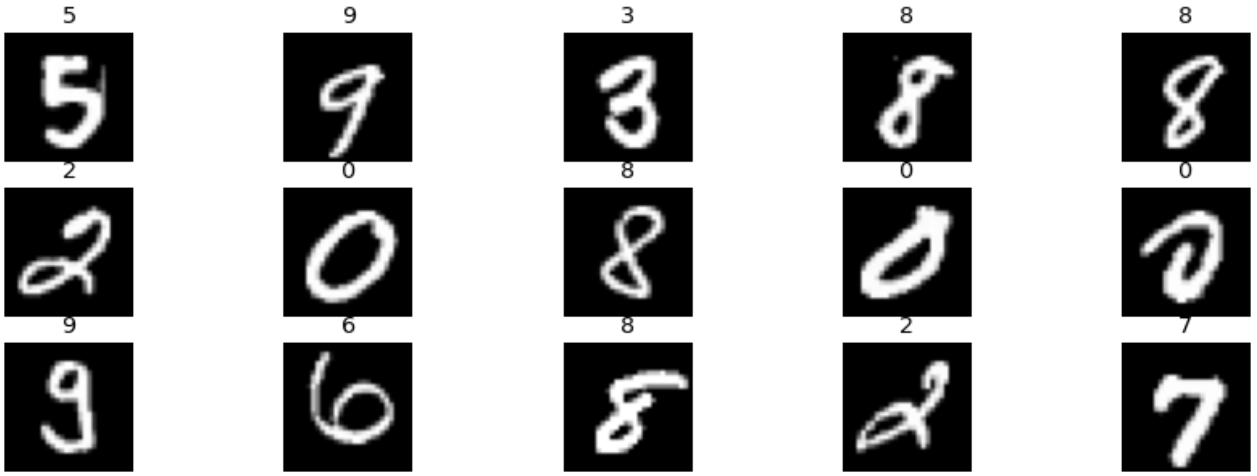
(http://yann.lecun.com/exdb/mnist/)

In [16]:
```
from keras.datasets import mnist

(images, labels), (images_test, labels_test) = mnist.load_data()

images = images/255.
images_test = images_test/255.
print(images.shape,labels.shape,images_test.shape,labels_test.shape)
```

```
(60000, 28, 28) (60000,) (10000, 28, 28) (10000,)
```

In [17]:
```
plt.figure(figsize=(15,5))
for i in range(15):
    idx = npr.choice(len(images),size=1)
    plt.subplot(3,5,i+1)
    plt.imshow(np.squeeze(images[idx,:,:],axis=0), cmap='gray')
```

```
    plt.title(labels[idx][0])
    plt.axis('off');
```



In [18]:

```
# Flattening image into vector
N,D,_ = images.shape
Ntest,D,_ = images_test.shape

X = images.flatten().reshape(N, D*D)
X_test = images_test.flatten().reshape(Ntest, D*D)

X.shape, X_test.shape
```

Out[18]:

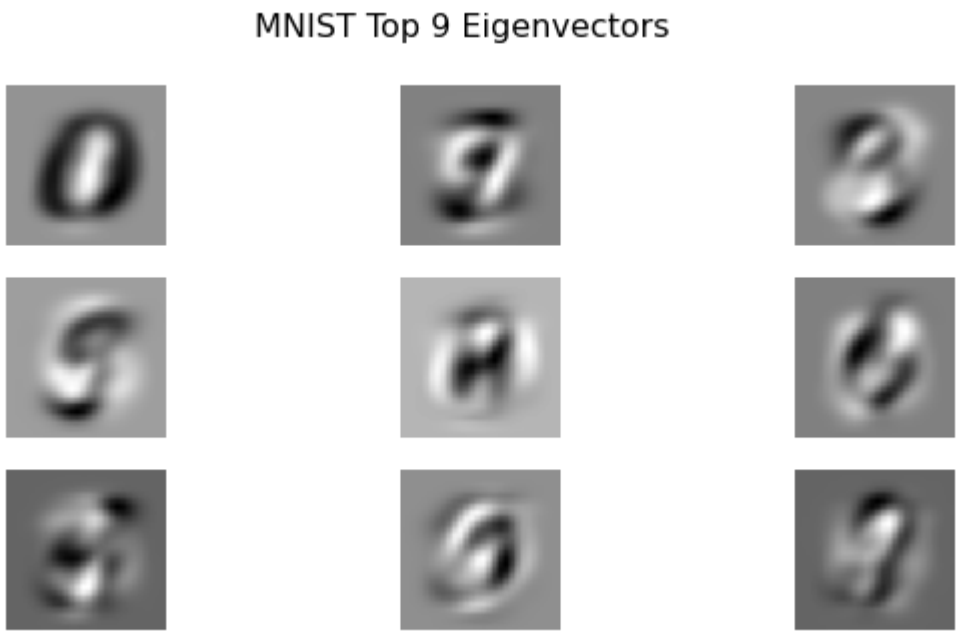
```
((60000, 784), (10000, 784))
```

In [19]:

```
A, eigen_pairs, cum_var_exp = PCA(X.T,2,0)

y = X@A
```

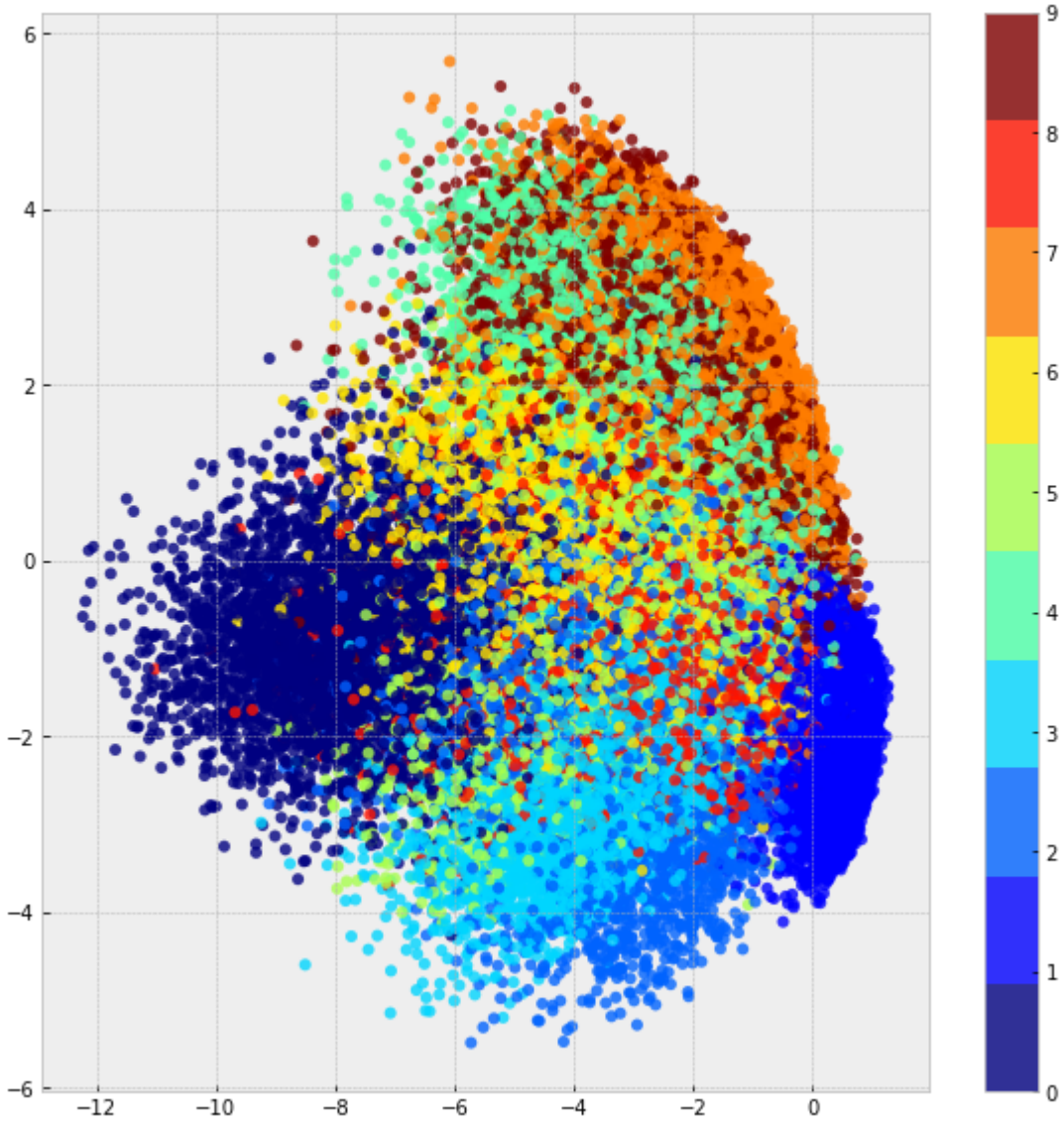
In [20]:

```
plt.figure(figsize=(10,5))
plt.suptitle('MNIST Top 9 Eigenvectors', size=16)
for i in range(9):
        plt.subplot(3, 3, i + 1)
        plt.imshow(np.real(eigen_pairs[i][1].reshape(28,28)),cmap='gray')
        plt.axis('off');
```



In [21]:

```
fig, ax = plt.subplots(figsize=(10, 10))
plt.scatter(np.real(y[:, 0]), np.real(y[:, 1]), c=labels, edgecolor='none', alpha=0.8,
plt.colorbar(ticks=range(10));
```



Recap

PCA is an **unsupervised** model that can be used to:

1. Perform Dimensionality Reduction, by projecting data into directions with maximum explained variance
2. Uncorrelate Data, by rotating the data space such that data becomes uncorrelated

PCA uses **linear transformations** (projections or rotations) of the input data X:

$$Y = AX$$

where A is a $D \times D$ linear transformation matrix, X is an $D \times N$ data matrix, and Y is a $D \times N$ transformed data matrix.

Therefore PCA will work well when the relationship between features are linear. Moreover, PCA is unsupervised because it **does not** use the class labels to find vector projections (or rotations). So the projections may not necessarily be in the direction that maximize class separability.

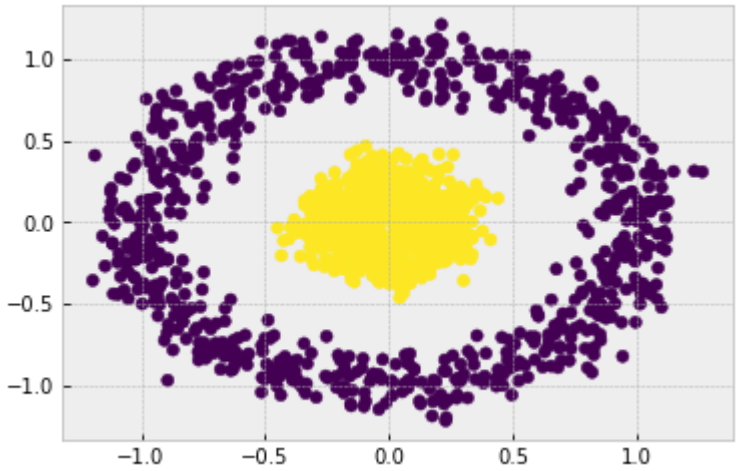
PCA can be formulated from two points-of-view:

1. Maximum explained variance
2. Minimum reconstruction error

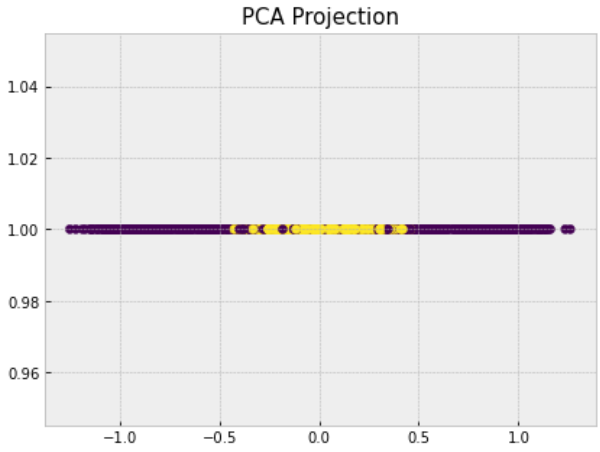
Kernel PCA

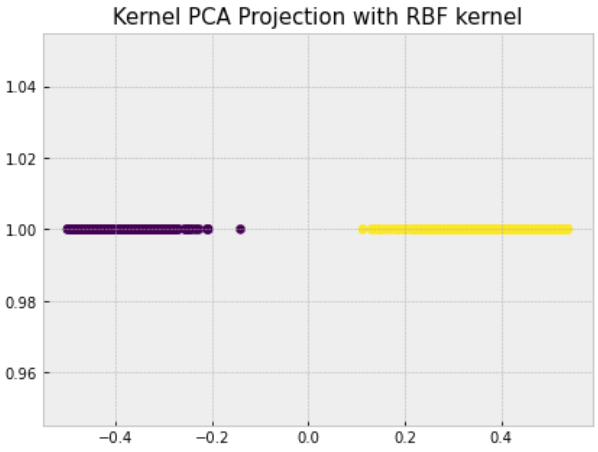
There are other variants of PCA such as **Kernel PCA**, where we first project the data to a space where classes are linearly separable (RBF kernel) and then apply PCA:

In [22]:
```
from sklearn.datasets import make_circles

X, y = make_circles(n_samples=1500, noise=0.1, factor=0.2)
plt.scatter(X[:,0],X[:,1],c=y);
```



In [23]:
```
from sklearn.decomposition import PCA, KernelPCA

model_pca = PCA(n_components=1)
proj = model_pca.fit_transform(X)

model_kpca = KernelPCA(n_components=1,kernel='rbf',gamma=2)
proj_kpca = model_kpca.fit_transform(X)

plt.figure(figsize=(15,5))
plt.subplot(121); plt.scatter(proj,np.ones(len(proj)),c=y)
plt.title('PCA Projection', size=15)
plt.subplot(122); plt.scatter(proj_kpca,np.ones(len(proj_kpca)),c=y)
plt.title('Kernel PCA Projection with RBF kernel',size=15);
```


PCA Projection
1.04
1.02
1.00
0.98
0.96
−1.0
−0.5
0.0
0.5
1.0


Kernel PCA Projection with RBF kernel
1.04
1.02
1.00
0.98
0.96
−0.4
−0.2
0.0
0.2
0.4