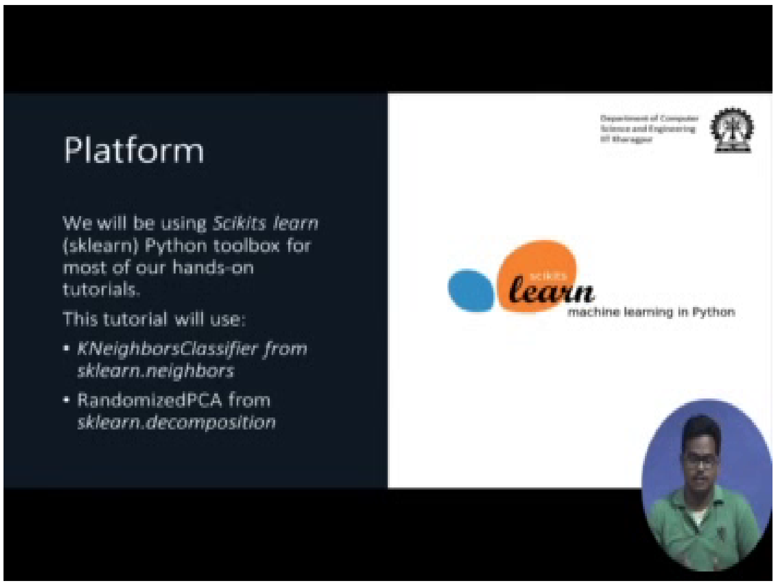
Introduction to Machine Learning
Prof. Anirban Santara
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 14
Python Exercise on kNN and PCA

Hello everyone, welcome to the second hands on session of the Introduction to Machine Learning course. I am Anirban Santara; I am doing my PhD in machine learning. Today we will learn couple of cool machine learning algorithms.

In the first part of the session, we will study how to use K-Nearest Neighbor classification algorithm for classification of flowers of the iris data set and in the second part we will learn how to use K-Nearest Neighbor classifier along with principle component analysis for face recognition.

(Refer Slide Time: 00:55)



Platform

We will be using *Scikits learn* (sklearn) Python toolbox for most of our hands-on tutorials.

This tutorial will use:

- *KNeighborsClassifier from sklearn.neighbors*
- RandomizedPCA from *sklearn.decomposition*

We will use python language as our language of choice for programming in this course. Scikits learn is a very popular machine learning library and it has a lot of machine learning utilities. We will study how to use K-Nearest Neighbor classifier and randomized PCA from Scikits learn in this session. Without further adieu let us jump into the exercises.

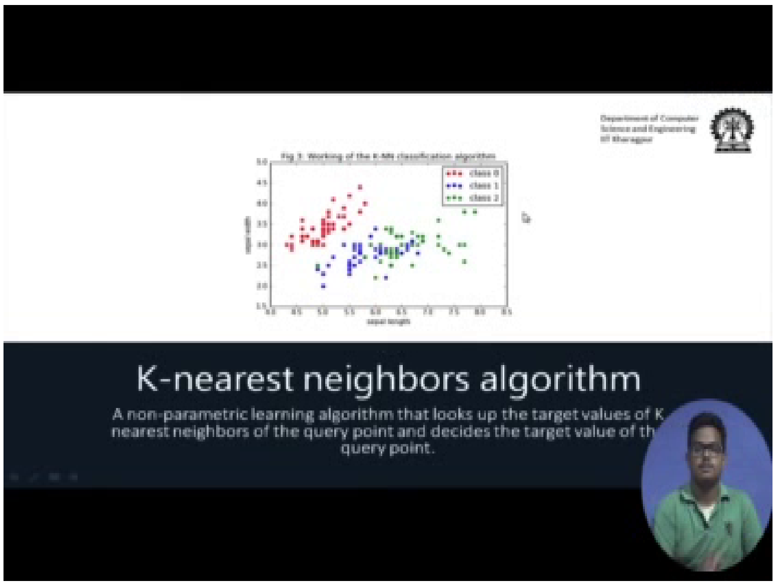
(Refer Slide Time: 01:21)

Department of Computer Science and Engineering IIT Kharagpur

K-Nearest Neighbors

The first topic for today is K-Nearest Neighbor classifier.

(Refer Slide Time: 01:25)



Department of Computer Science and Engineering IIT Kharagpur

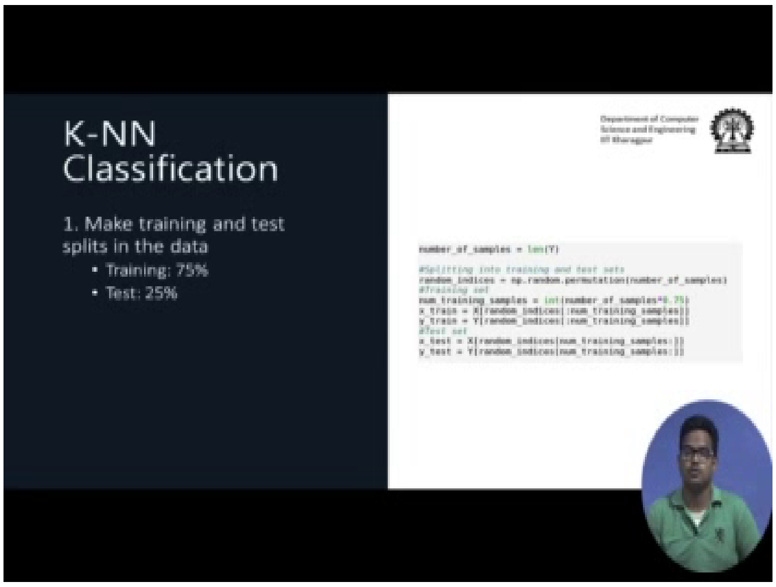
K-nearest neighbors algorithm

A non-parametric learning algorithm that looks up the target values of K nearest neighbors of the query point and decides the target value of th query point.

The k nearest neighbor algorithm is a non-parametric machine learning algorithm and it maintains a database of the training samples and it every time a query is made to the algorithm it looks up the database and it finds the K, which is specified by the user nearest neighbors of the query point from the data base.

Now, once it has retrieve this K-Nearest Neighbor it goes and finds out which class is the most predominant among the retrieved neighbors all right and the most predominant class is assigned as a target class for the query point and we will use a modified version of the iris data set in this exercise. The iris data set has flowers and these iris flowers fall in three species all right and task is to classify the iris flower from the sepal and petal dimension for the ease of visualization, we will chose just a first two feature dimensions that is the sepal dimension, sepal length and the sepal width for describing the iris flowers.

(Refer Slide Time: 02:47)



K-NN Classification

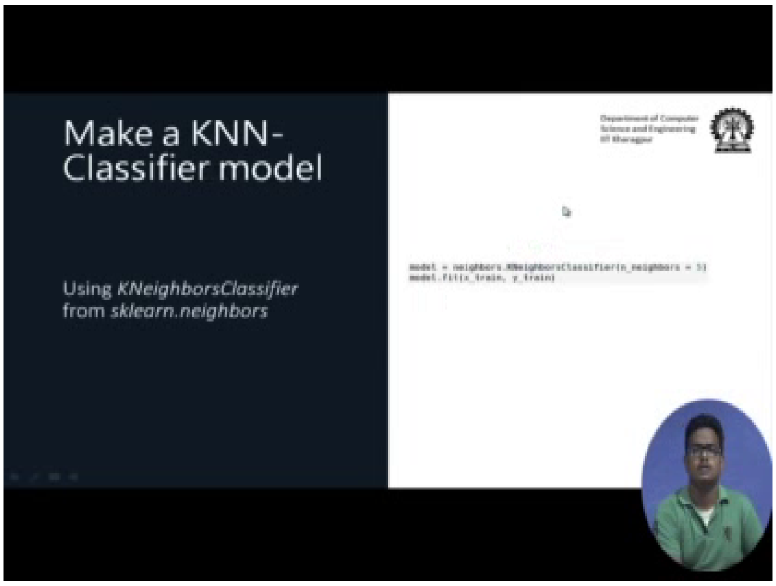
1. Make training and test splits in the data
    - Training: 75%
    - Test: 25%

Department of Computer Science and Engineering IIT Kharagpur

```
number_of_samples = len(Y)

#Splitting into training and test sets
random_indices = np.random.permutation(number_of_samples)
#Training set
num_training_samples = int(number_of_samples*0.75)
x_train = X[random_indices[:num_training_samples]]
y_train = Y[random_indices[:num_training_samples]]
#Test set
x_test = X[random_indices[num_training_samples:]]
y_test = Y[random_indices[num_training_samples:]]
```

The first task is to split the available data into training and test seconds. So, we randomly chose 75 percent of the data that is available to us for our training set and save the remaining 25 percent for the test set. So, this part of the code which you see on the screen describes how to do it in code.

(Refer Slide Time: 03:12)



Make a KNN-Classifier model

Using *KNeighborsClassifier* from *sklearn.neighbors*

Department of Computer Science and Engineering
IIT Kharagpur

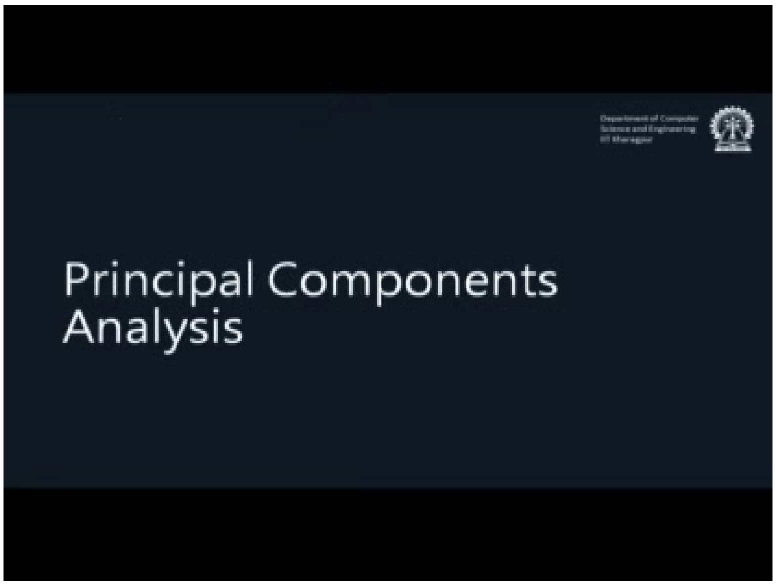
```
model = neighbors.KNeighborsClassifier(n_neighbors = 5)
model.fit(x_train, y_train)
```

The next step is to go ahead and make the k-nearest neighbor classifier. So, for that you have to make an object of the class k-neighbors classifier and you have to set the numbers of neighbors to our required value. So, for example, I have set it equal to 5 and then we do a module dot fit x train and y train. So, this thing loads the data set into the module and saves it for reference when query is made. The next step is to check out how the algorithm is performing. So, we find the query point and the query point is first example from the test set.

So, you get a prediction from the module by using module dot predict of the query point and in this way. So, the nearest neighbors module of the scikit learn dot neighbors library users helps us to visualize how the algorithm actually works out all right.

So, here you can see the query point is the dark blue triangle and the neighbors have been highlighted in yellow and we can see that class two is the most predominant among the classes of the neighbors and hence it predicts the class two as a class of the query point all right and so this is how the nearest neighbor algorithm works and it is highly popular algorithm and it works well when say the data set is varying in its number of classes, for example, you cannot afford to train a parametric machine learning module time and again. Every time the data changes of the number of classes that you want you know you want your classifier to predict that changes with time. So, the K-nearest neighbor algorithm becomes highly relevant in those scenarios.

(Refer Slide Time: 05:07)



Principal Components Analysis

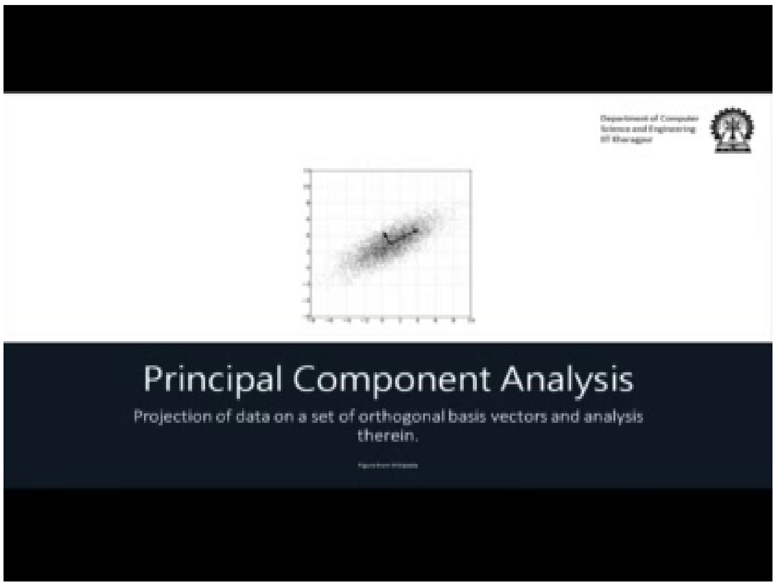
Department of Computer Science and Engineering IIT Kharagpur

The next part of the session is about principal component analysis. So, the principal component analysis algorithm is a highly popular algorithm used for dimensionality reduction. Now, dimensionality reduction means reducing the number of variables that we use for representing our data. Now, why does it become relevant? In some cases, it becomes highly relevant we have to deal with high dimensional inputs spaces. So, our phenomenon called curves of dimensionality happens in such scenario.

So, curves of dimensionality actually refers to a set of problems that crop up when we have too many dimensions, and this problem crop up just because the volume of the features space increases at such a high rate as the number of feature dimension increases that the amount of data that is available to us becomes spares in that. In such high dimensional space and the amount of training examples that we would need for getting statistically sound results from a machine learning algorithm.

In such high dimensional vector spaces increases exponentially with the number of dimension of the input space, so that is why we would like to reduce the dimensionality of the features space without using much information, and then principal component analysis algorithm is one of those algorithms which become highly relevant in this particular scenario.

(Refer Slide Time: 06:41)



Department of Computer Science and Engineering IIT Kharagpur

Principal Component Analysis

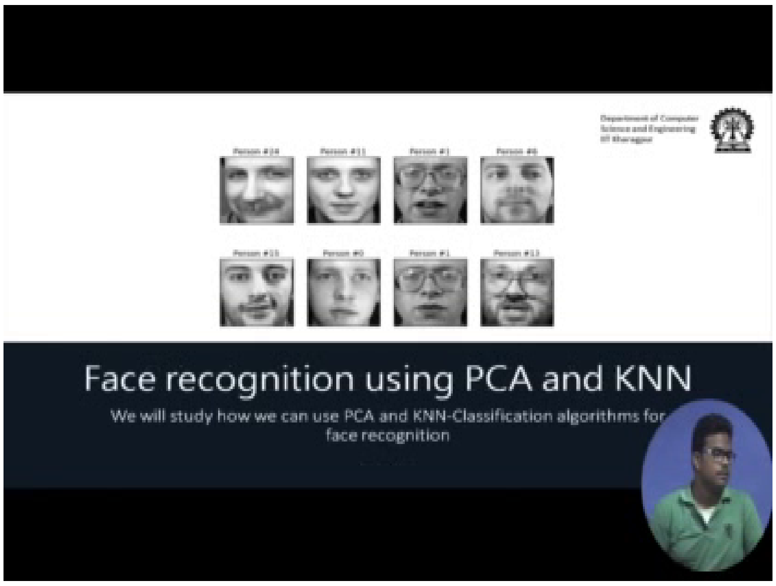
Projection of data on a set of orthogonal basis vectors and analysis therein.

Figure from Wikipedia

So, the principal component analysis algorithm it looks for uncorrelated features dimensions all right. Sometimes what happens is the observation we make the variables that we observe are highly correlated among themselves, for example, in this diagram you can see that the x and y axis, which the data observe data originally has or highly correlated right and the principal component analysis instead chooses a pair of different directions which are mutually orthogonal to each other. And if we represent the data along like just one of these principal components were the most of the variance of the data is preserved and the principal component analysis algorithm is looks for uncorrelated feature dimension.

So, the amount of redundancy that is present within the correlated variables, it is removed when we transform the data into the space of the principal components and the principal component happen to be the Eigen vectors of the covariance matrix of the data, and let us study the principal component analysis algorithm and how it works and how we can use it to reduce the dimensionality of images of faces and use the reduce dimensional face vector for face recognition.

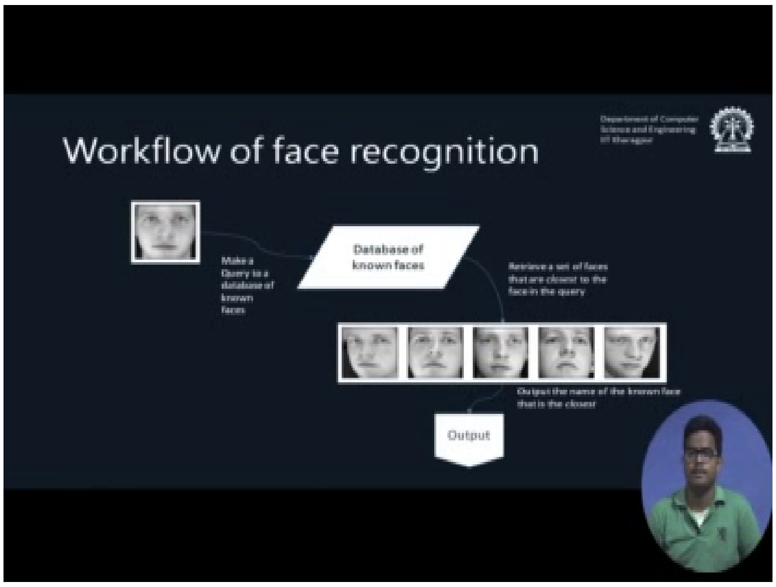
(Refer Slide Time: 08:03)



Face recognition using PCA and KNN

We will study how we can use PCA and KNN-Classification algorithms for face recognition

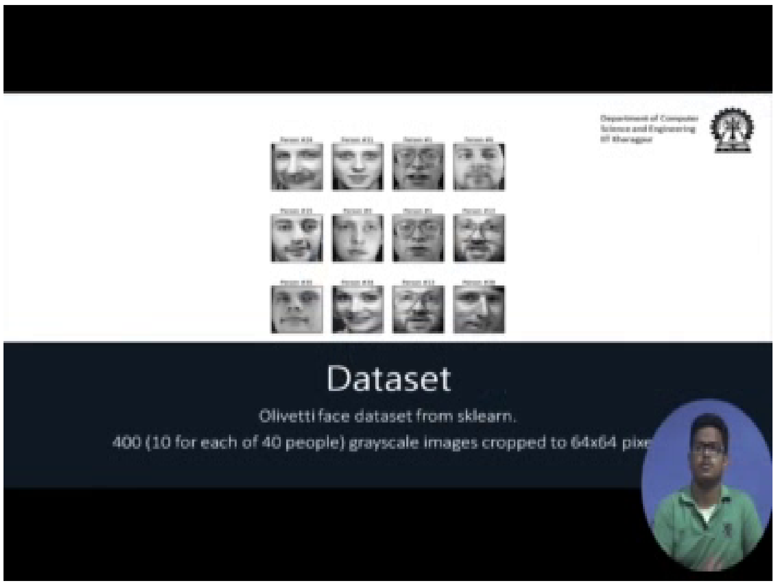
(Refer Slide Time: 08:08)


Department of Computer Science and Engineering IIT Kharagpur
Make a Query to a database of known faces
Database of known faces
Retrieve a set of faces that are closest to the face in the query
Output the name of the known face that is the closest
Output

Workflow of face recognition

The general work flow of any face recognition algorithm goes as follows. First we have a whole data base of known faces and once a new face is presented we go and make a query to the database. The database returns a set of faces which look the closest to the face in the query and all of these faces are know all right. So, the database has known faces. So, it outputs the set of faces which are already known to us and are the closest to the one that has been sent in the query and then we go ahead and chose the face that is the most similar to the face in the query as our output. So, this is the work flow.

(Refer Slide Time: 08:57)

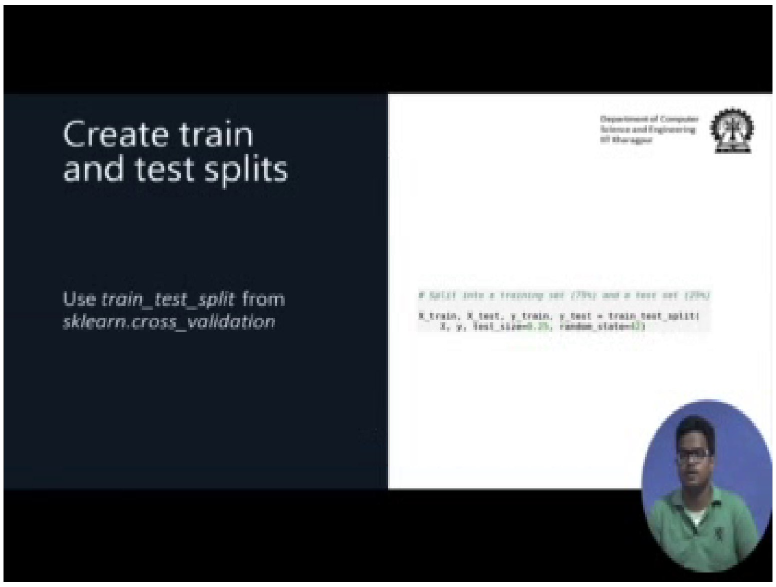

Dataset

Olivetti face dataset from sklearn.

400 (10 for each of 40 people) grayscale images cropped to 64x64 pixe

Now, how does principal component analysis work and in this particular you know scenario we will study in this section of the tutorial. The data set that we will be concerned with is the Olivetti face data set of Scikit learn. These are faces grayscale images of faces of forty people and each of these forty people have ten faces altogether. We have 400 faces and these faces have been cropped to a size of 64 cross 64 pixels.

(Refer Slide Time: 09:29)

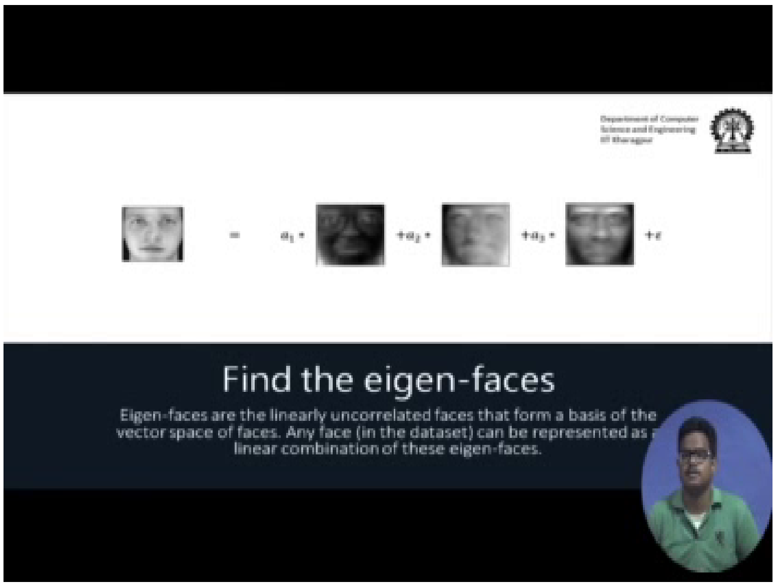

Create train and test splits

Use *train_test_split* from *sklearn.cross_validation*

```
# Split into a training set (75%) and a test set (25%)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42)
```

And next we go ahead and make our train and test split 75 percent training data, 25 percent test data and we use the train test split function of sklearn dot cross validation for doing this exercise.

(Refer Slide Time: 09:48)


Department of Computer Science and Engineering IIT Kharagpur
= a_1 * +a_2 * +a_3 * +ε

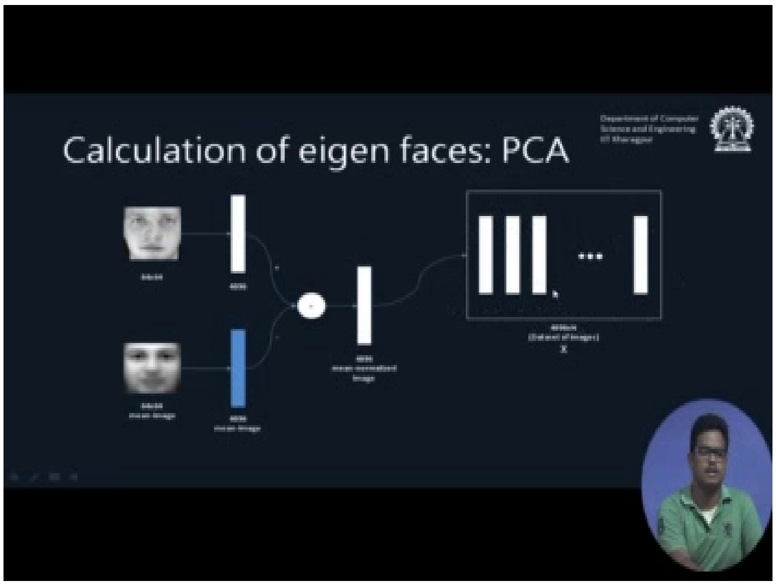
Find the eigen-faces

Eigen-faces are the linearly uncorrelated faces that form a basis of the vector space of faces. Any face (in the dataset) can be represented as a linear combination of these eigen-faces.

Then we go ahead and find the eigen-faces. So, what are eigen-faces? Eigen-faces are the principal component that we were talking about the eigen-faces. They are the eigen vectors of the covariance matrix of the data set of faces. So, these eigen-faces are a set of components, a set of vectors you can say and if a face which can you know which encode a lot of information about faces.

So, a face can be represented as a linear combination of these eigen-faces and say we have face of 64 cross 64 dimensions and that makes 4096 random variable for each pixel of the face. Now, instead of 4096 random variables which are highly correlated among themselves, we will go ahead and represent the faces in terms of the principal component and we will be using just 150 of them. Thus we have done a huge dimensional reduction.

(Refer Slide Time: 10:47)



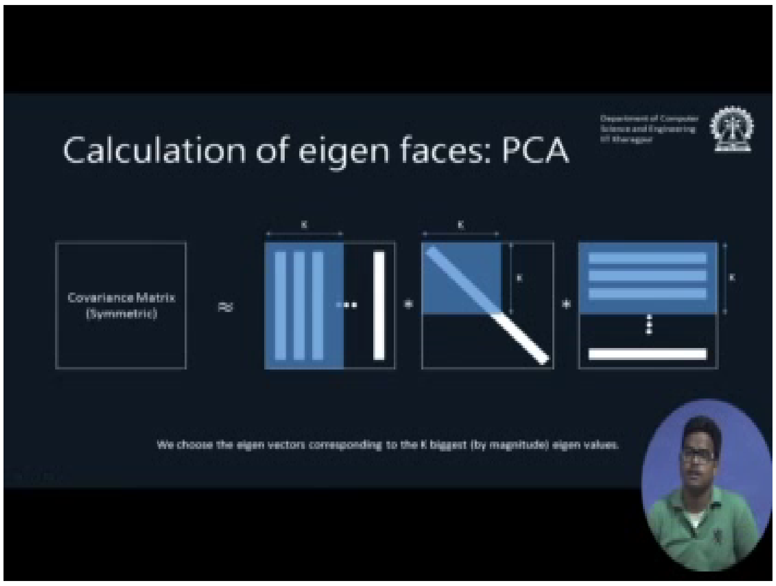
Calculation of eigen faces: PCA

Department of Computer Science and Engineering IIT Kharagpur
64x64
4096
64x64 mean image
4096 mean image
4096 mean normalized image
4096xN (Dataset of images) X

And next, we will see how the principal component analysis algorithm works. So, given face 64, first we go ahead and make a vector out of that. So, we just do a raster scan on the image and we arrange the pixels along the rows into a vector of 4096 dimensions and next we go ahead and find the mean vector and suppose and subtract from the original image. So, the mean face has been calculated by the averaging all the faces in the data in the training data set and that is also converted into vector and subtracted from the image.

And last we have the mean normalized image. This is necessary because it helps in optimization algorithm of optimization of any gradient descent based algorithm down the road and it is also necessary for the calculation of the covariance matrix because the definition of covariance is actually, expected value of x minus mu x times x minus mu x transpose which has been explained to in the theory section. And thus we do this exercise for all the images of the training set and we have the 4096 dimensional mean normalized images and thus we go ahead and create a matrix called x and each column of the matrix is one image from the training set. So, this x is a set of mean normalized training images, say we have capital n training images and thus the size is of the matrix x we will be 4096 cross n.

(Refer Slide Time: 12:27)


Calculation of eigen faces: PCA

We choose the eigen vectors corresponding to the K biggest (by magnitude) eigen values.

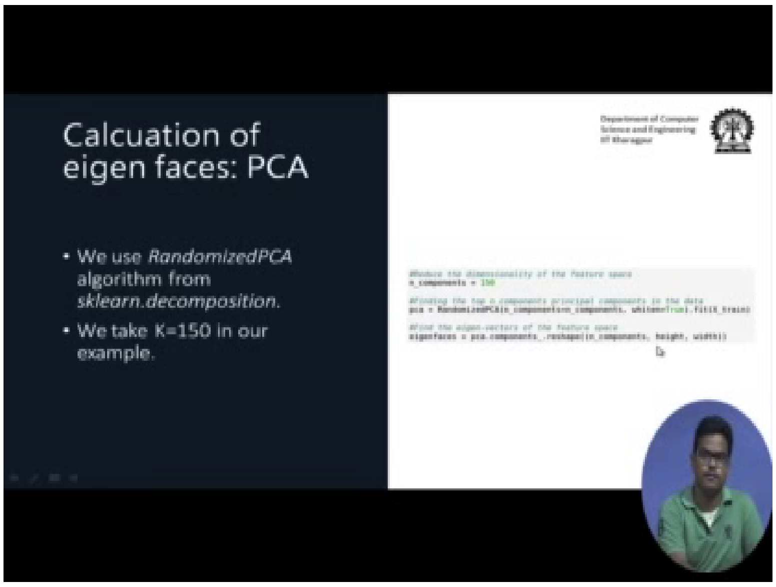
Next, we go ahead and calculate the covariance matrix of the data. So, we multiply x with the transpose of x and we have r, the covariance matrix. The dimensionality of covariance matrix is 4096 cross 4096. The principal component will be the eigen vectors of the covariance matrix and we will see next how to calculate the eigen vectors.

So, we will calculate the eigen vectors by using a procedure which is called diagonalisation in the context of symmetric metrics and in general in is called singular validly composition. So, what does it do? It represents the matrix r as a product of 3 metrics, the first one is called p and it consists of the eigen vectors along the columns. The last one is the same matrix transposed and the central one is a diagonal matrix each element along the diagonal i and eigen value and it corresponds to the eigen vector in the matrix p. So, the first i column of p is an eigen vector the eigen value of which is the first element of d along the diagonal all right. So, what we will do is we will just use the eigen vectors corresponding to the highest magnitude eigen values.

So, the k the magnitude of the eigen value shows which is how much a one particular eigen vector contributional to the information; how much information a particular eigen vector carries about the data all right. So, we chose the k highgon highest eigen values and thier corresponding eigen vectors. So, that the. So, that maximum information is preserved all right. So, we just chose the first k columns because these k columns are corresponding to highest magnitude eigen vector eigen value.

So, we should first make sure that the eigen values are arranged in decreasing order of the magnitudes and corresponding eigen vectors are present in the same you know same column location as the value as the eigen value all right. So, we just chose the top k eigen values, and eigen vectors and these this value of k is what we specify to the principal component analysis algorithm and we specify that to be 150. So, know we will do this exercise this part of the exercise in code.

(Refer Slide Time: 15:06)



Calcuation of eigen faces: PCA

- We use *RandomizedPCA* algorithm from *sklearn.decomposition*.
- We take K=150 in our example.

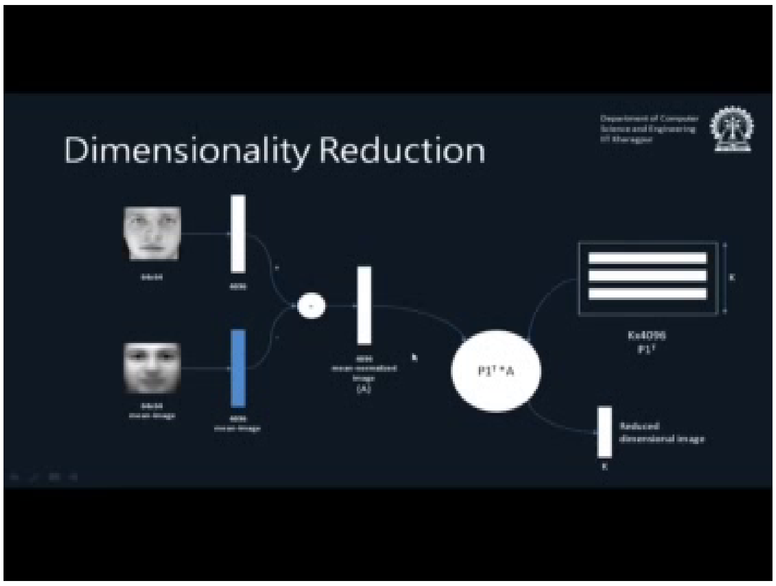
Department of Computer Science and Engineering IIT Kharagpur

```
#Reduce the dimensionality of the feature space
n_components = 150

#Finding the top n components principal components in the data
pca = RandomizedPCA(n_components=n_components, whiten=True).fit(X_train)

#Find the eigen-vectors of the feature space
eigenfaces = pca.components_.reshape((n_components, height, width))
```

So, the randomize PCA algorithm of Scikit learn it does efficient incrementation of the components as algorithms. So, we said we first specify that number of components will be 150 and then we initialize PCA model and we say that the number of component is the equal to 150 and we fit on the remaining data, this all the operation that we saw. Secondly, showed before the entire mean normalization the SVD and then we tangiest try and visualize the eigen faces.

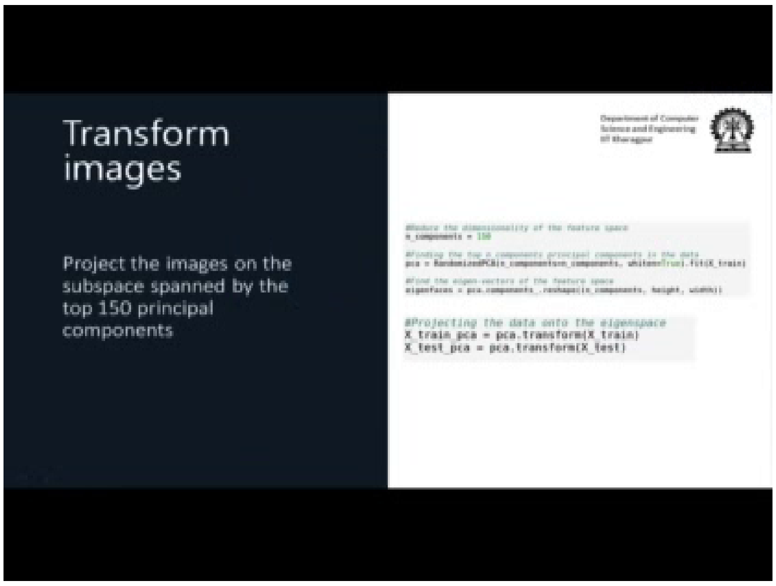
So, eigen faces are calculated eigen vectors are calculated they are store within the model. We can retrieve them as pca dot component s underscore all right and then we reshape them to height width. So, that we can visualize them well and we will show them in a minute. So, once we have identified the principal components we go ahead and do our dimensionality reduction, we just preserve the parts that are necessary to us just a first cake principal components.

(Refer Slide Time: 16:07)



And as you can see the dimensionality is now 4096 cross k and we have changed the names to the previous names followed by 1 just to avoid ambiguity. So, now, we do the dimensionality reduction. So, what do we do given an input image we first mean normalize, it make the find the vector we us the mean image as we calculated before and we subtract the mean image and we calculate the mean normalized image. Now, the mean normalized image is transformed with p 1 transpose, we multiply it p 1 transpose into a, the mean normalized image and what we have is a k dimensional vector which represent the image in reduce dimensional space span by the top k eigen vectors. So, this is how the dimensionality reduction works using PCA.

(Refer Slide Time: 17:02)



Transform images

Project the images on the subspace spanned by the top 150 principal components

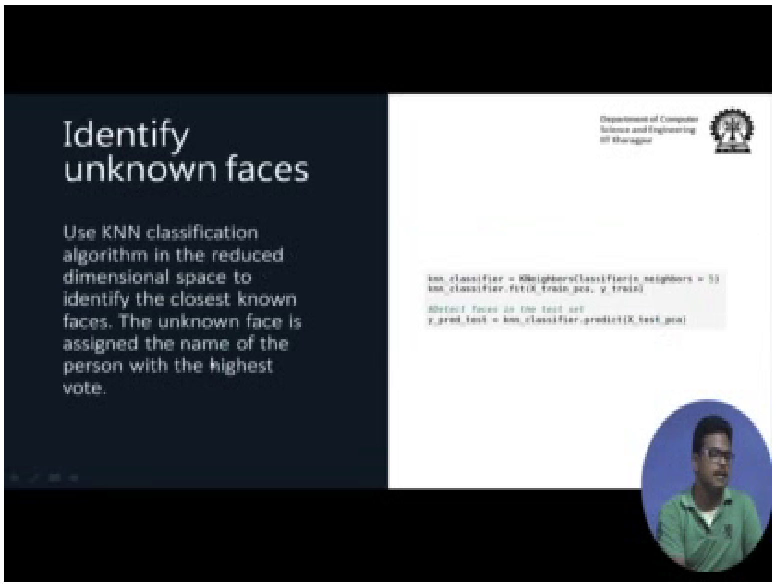
Department of Computer Science and Engineering IIT Kharagpur

```
#Reduce the dimensionality of the feature space
n_components = 150

#Finding the top n_components principal components in the data
pca = RandomizedPCA(n_components=n_components, whiten=True).fit(X_train)

#Find the eigen-vectors of the feature space
eigenfaces = pca.components_.reshape((n_components, height, width))
```

```
#Projecting the data onto the eigenspace
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
```

And now we will use the now we will implement that in the code first. So, pca dot transform of a set of input vectors. We will do the transformation to the PCA dimensions, this is what we do here, and we transform both the trainee set and the test set into 150 principal components.

(Refer Slide Time: 17:22)



Identify unknown faces

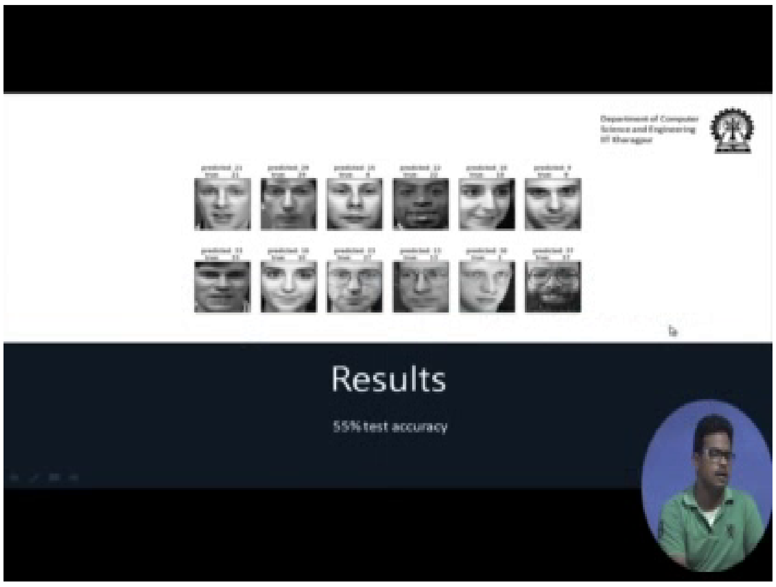
Use KNN classification algorithm in the reduced dimensional space to identify the closest known faces. The unknown face is assigned the name of the person with the highest vote.

Department of Computer Science and Engineering IIT Kharagpur

```
knn_classifier = KNeighborsClassifier(n_neighbors = 5)
knn_classifier.fit(X_train_pca, y_train)

#Detect faces in the test set
y_pred_test = knn_classifier.predict(X_test_pca)
```

And then we do k-nearest neighbor classification of faces in this principal component vector space, in the space span by the principal components rather. So, what do we do we first declare our classifier we fit it on the trainee set and then we do a classification on the test set.

(Refer Slide Time: 17:47)



Department of Computer Science and Engineering IIT Kharagpur

Results

55% test accuracy

And the results are quite nice quite impressive and as you can see some examples of classification and we have 55 percent test accuracy with numbers of neighbors equal to 5 and we can further tune the number of neighbors and have better and better accuracy, so that was nice demonstration of how principle components analysis and k-nearest neighbor algorithm can come together and do a solve a very interesting and very cool problem like face recognition.

Thank you guys see you in the next video. Bye-Bye.