ML_miniProject

June 4, 2021

[1]:
```
# get the dataset
!git clone https://github.com/Horea94/Fruit-Images-Dataset
```

```
Cloning into 'Fruit-Images-Dataset'…
remote: Enumerating objects: 385858, done.
remote: Counting objects: 100% (8693/8693), done.
remote: Compressing objects: 100% (8672/8672), done.
remote: Total 385858 (delta 36), reused 8670 (delta 21), pack-reused 377165
Receiving objects: 100% (385858/385858), 2.10 GiB | 23.10 MiB/s, done.
Resolving deltas: 100% (1196/1196), done.
Checking out files: 100% (90503/90503), done.
```

[2]:
```
# import
import numpy as np
import cv2
import glob
import os
import matplotlib.pyplot as plt
import string
from mlxtend.plotting import plot_decision_regions
from mpl_toolkits.mplot3d import Axes3D
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.utils.multiclass import unique_labels
from sklearn import metrics
from sklearn.svm import SVC

print(os.listdir("/content/"))
dim = 100
```

```
['.config', 'Fruit-Images-Dataset', 'sample_data']
```

[3]:
```
def getDataset(fruits, data_type, print_n=False, k_fold=False):
    '''
    loads the dataset and labels
```

```
    :params:
     fruit     : the fruits to load
     data_type : train/test data
     print_n   : print the steps or not
     k_fold    : perform K-fold cross validation

    :returns
     images : loaded images
     labels : corresponding labels of images
    '''
    images = []
    labels = []
    val = ['Training', 'Test']
    if not k_fold:
        path = "/content/Fruit-Images-Dataset/" + data_type + "/"
        for i,f in enumerate(fruits):
            p = path + f
            j=0
            for image_path in glob.glob(os.path.join(p, "*.jpg")):
                image = cv2.imread(image_path, cv2.IMREAD_COLOR)
                image = cv2.resize(image, (dim, dim))
                image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
                images.append(image)
                labels.append(i)
                j+=1
            if(print_n):
                print("There are " , j , " " , data_type.upper(), " images of "␣
↪, fruits[i].upper())
        images = np.array(images)
        labels = np.array(labels)
        return images, labels
    else:
        for v in val:
            path = "/content/Fruit-Images-Dataset/" + v + "/"
            for i,f in enumerate(fruits):
                p = path + f
                j=0
                for image_path in glob.glob(os.path.join(p, "*.jpg")):
                    image = cv2.imread(image_path, cv2.IMREAD_COLOR)
                    image = cv2.resize(image, (dim, dim))
                    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
                    images.append(image)
                    labels.append(i)
                    j+=1
        images = np.array(images)
        labels = np.array(labels)
        return images, labels
```

[4]:
```python
def getAllLabels():
    '''
    gets all the available labels in the dataset
    '''
    fruits = []
    for fruit_path in glob.glob("/content/Fruit-Images-Dataset/Training/*"):
        fruit = fruit_path.split("/")[-1]
        fruits.append(fruit)
    return fruits
```

[5]:
```python
#Choose Fruits
fruits = ['Onion Red' , 'Fig']

#Get Images and Labels
X_t, y_train =  getDataset(fruits, 'Training', print_n=True, k_fold=False)
X_test, y_test = getDataset(fruits, 'Test', print_n=True, k_fold=False)

#Get data for k-fold
X,y = getDataset(fruits, '', print_n=True, k_fold=True)

#Scale Data Images
scaler = StandardScaler()
X_train = scaler.fit_transform([i.flatten() for i in X_t])
X_test = scaler.fit_transform([i.flatten() for i in X_test])
X = scaler.fit_transform([i.flatten() for i in X])
```

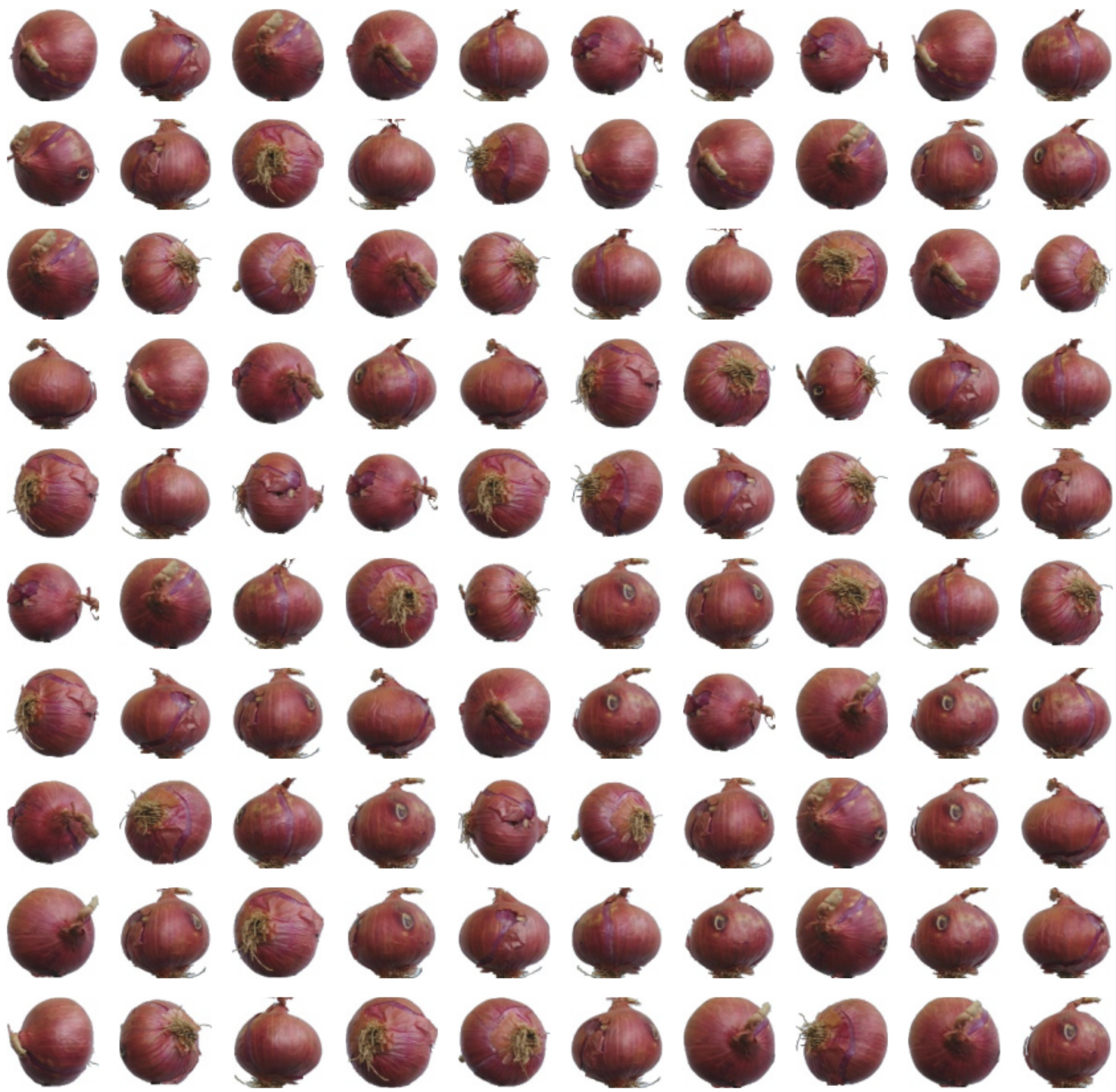
```
There are  450   TRAINING  images of  ONION RED
There are  702   TRAINING  images of  FIG
There are  150   TEST  images of  ONION RED
There are  234   TEST  images of  FIG
```

[6]:
```python
def plot_image_grid(images, nb_rows, nb_cols, figsize=(15, 15)):
    '''
    plots sample images from the loaded dataset

    :params:
     images  : images to plot
     nb_rows : number of rows to display
     nb_col  : number of  columns to display
     figsize : size of the figure

    '''
    assert len(images) == nb_rows*nb_cols, "Number of images should be the same␣
↪as (nb_rows*nb_cols)"
    fig, axs = plt.subplots(nb_rows, nb_cols, figsize=figsize)

    n = 0
```

```python
    for i in range(0, nb_rows):
        for j in range(0, nb_cols):
            axs[i, j].axis('off')
            axs[i, j].imshow(images[n])
            n += 1
```

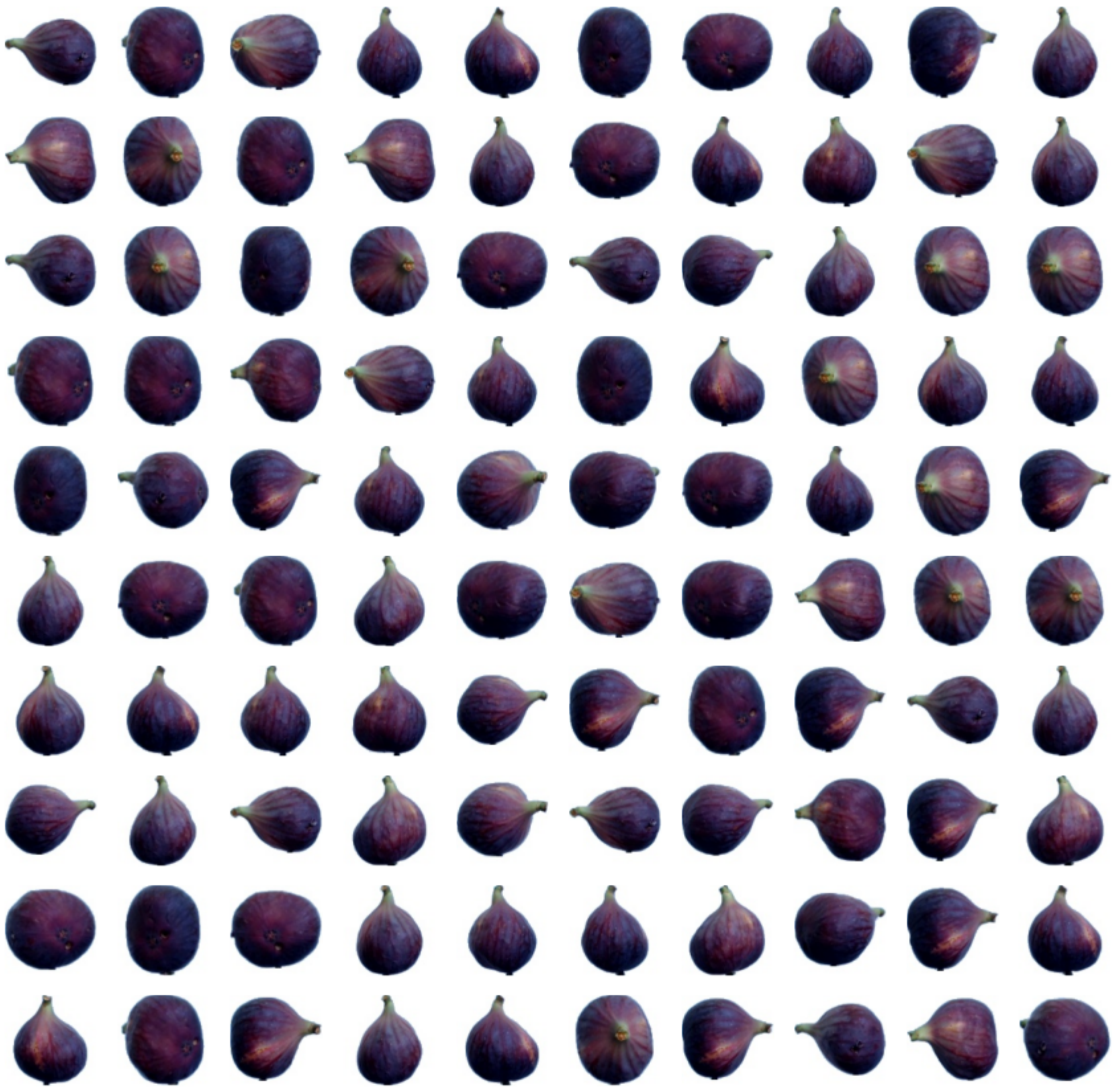
[7]:
```python
print(fruits[y_train[0]])
plot_image_grid(X_t[0:100], 10, 10)
```

```
Onion Red
```



[8]:
```python
print(fruits[y_train[451]])
plot_image_grid(X_t[451:551], 10, 10)
```

```
Fig
```



[9]:
```python
def getClassNumber(y):
    '''
    gets the count of each class

    :params:
     y : labels

    :returns
     v : count of each label
    '''
    v =[]
    i=0
```

```python
    count = 0
    for index in y:
        if(index == i):
            count +=1
        else:
            v.append(count)
            count = 1
            i +=1
    v.append(count)
    return v
```

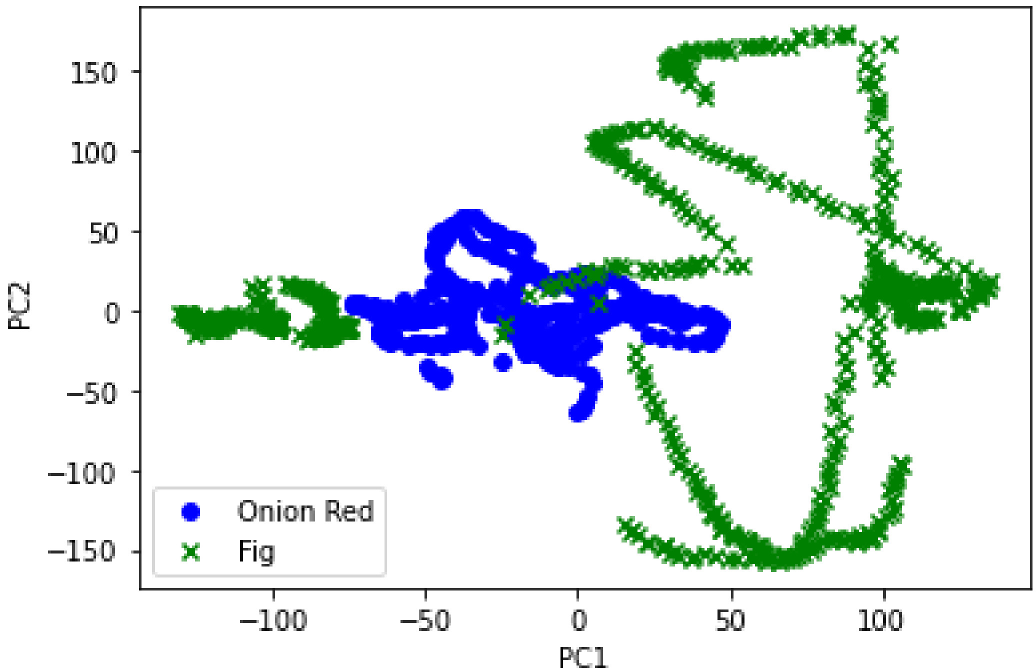
[10]:

```python
def plotPrincipalComponents(X, dim):
    '''
    plots PCs

    :params:
     X   : features
     dim : dimensions
    '''
    v = getClassNumber(y_train)
    colors = 'b', 'g', 'r', 'c', 'm', 'y', 'k', 'grey', 'orange', 'purple'
    markers = ['o', 'x' , 'v', 'd']
    tot = len(X)
    start = 0
    if(dim == 2):
        for i,index in enumerate(v):
            end = start + index
            plt.scatter(X[start:end,0],X[start:end,1] ,␣
↪color=colors[i%len(colors)], marker=markers[i%len(markers)], label =␣
↪fruits[i])
            start = end
        plt.xlabel('PC1')
        plt.ylabel('PC2')

    if(dim == 3):
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        for i,index in enumerate(v):
            end = start + index
            ax.scatter(X[start:end,0], X[start:end,1], X[start:end,2],␣
↪color=colors[i%len(colors)], marker=markers[i%len(markers)], label =␣
↪fruits[i])
            start = end
        ax.set_xlabel('PC1')
        ax.set_ylabel('PC2')
        ax.set_zlabel('PC3')
```

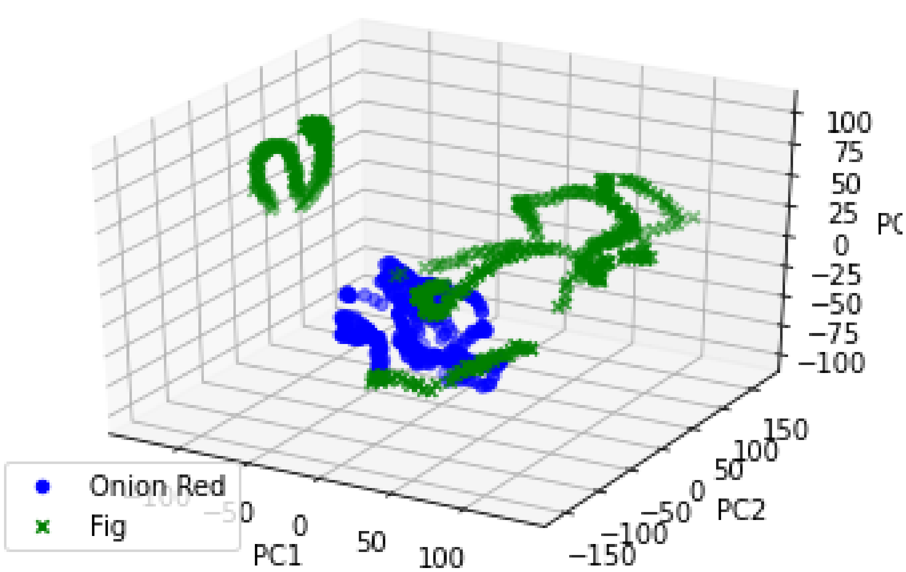
```
    plt.legend(loc='lower left')
    plt.xticks()
    plt.yticks()
    plt.show()
```

[11]:
```
def plot_confusion_matrix(y_true, y_pred, classes, normalize=False, title=None, cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = metrics.confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = unique_labels(y_true, y_pred)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]), yticks=np.arange(cm.shape[0]), xticklabels=fruits, yticklabels=fruits, title=title, ylabel='True label', xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt), ha="center", va="center", color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return cm,ax
```

[12]:
```
pca = PCA(n_components=2)
dataIn2D = pca.fit_transform(X_train)
plotPrincipalComponents(dataIn2D, 2)
```



[13]:
```
pca = PCA(n_components=3)
dataIn3D = pca.fit_transform(X_train)
plotPrincipalComponents(dataIn3D, 3)
```



[14]:
```python
def showPCA(image,X2, X10, X50):
    fig = plt.figure(figsize=(15,15))
    ax1 = fig.add_subplot(1,4,1)
    ax1.axis('off')
    ax1.set_title('Original image')
    plt.imshow(image)
    ax1 = fig.add_subplot(1,4,2)
    ax1.axis('off')
    ax1.set_title('50 PC')
    plt.imshow(X50)
    ax1 = fig.add_subplot(1,4,3)
    ax1.axis('off')
    ax1.set_title('10 PC')
    plt.imshow(X10)
    ax2 = fig.add_subplot(1,4,4)
    ax2.axis('off')
    ax2.set_title('2 PC')
    plt.imshow(X2)
    plt.show()
```

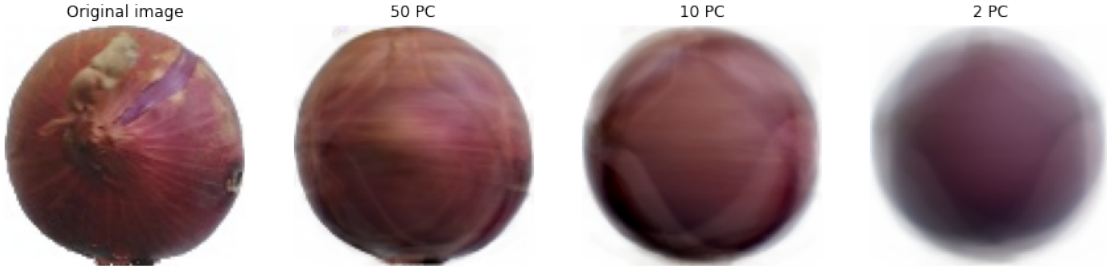
[15]:
```python
def computePCA(n, im_scaled, image_id):
    pca = PCA(n)
    principalComponents = pca.fit_transform(im_scaled)
    im_reduced = pca.inverse_transform(principalComponents)
    newImage = scaler.inverse_transform(im_reduced[image_id])
    return newImage
```

[16]:
```python
def showVariance(X_train):
    #Compute manually the principal components
    cov_matr=np.dot(X_train, X_train.T)
    eigval,eigvect=np.linalg.eig(cov_matr)

    index=np.argsort(eigval)[::-1] #take in order the index of ordered vector␣
 ↪(ascending order)

    #eigvect[:,i] is associated to eigval[i] so
    eigvect=eigvect[:,index]
    eigval=eigval[index]

    n_PC=[]
    var_explained=[]
    var_temp=[]
    var_tmp=0
    for i in range(10):
```

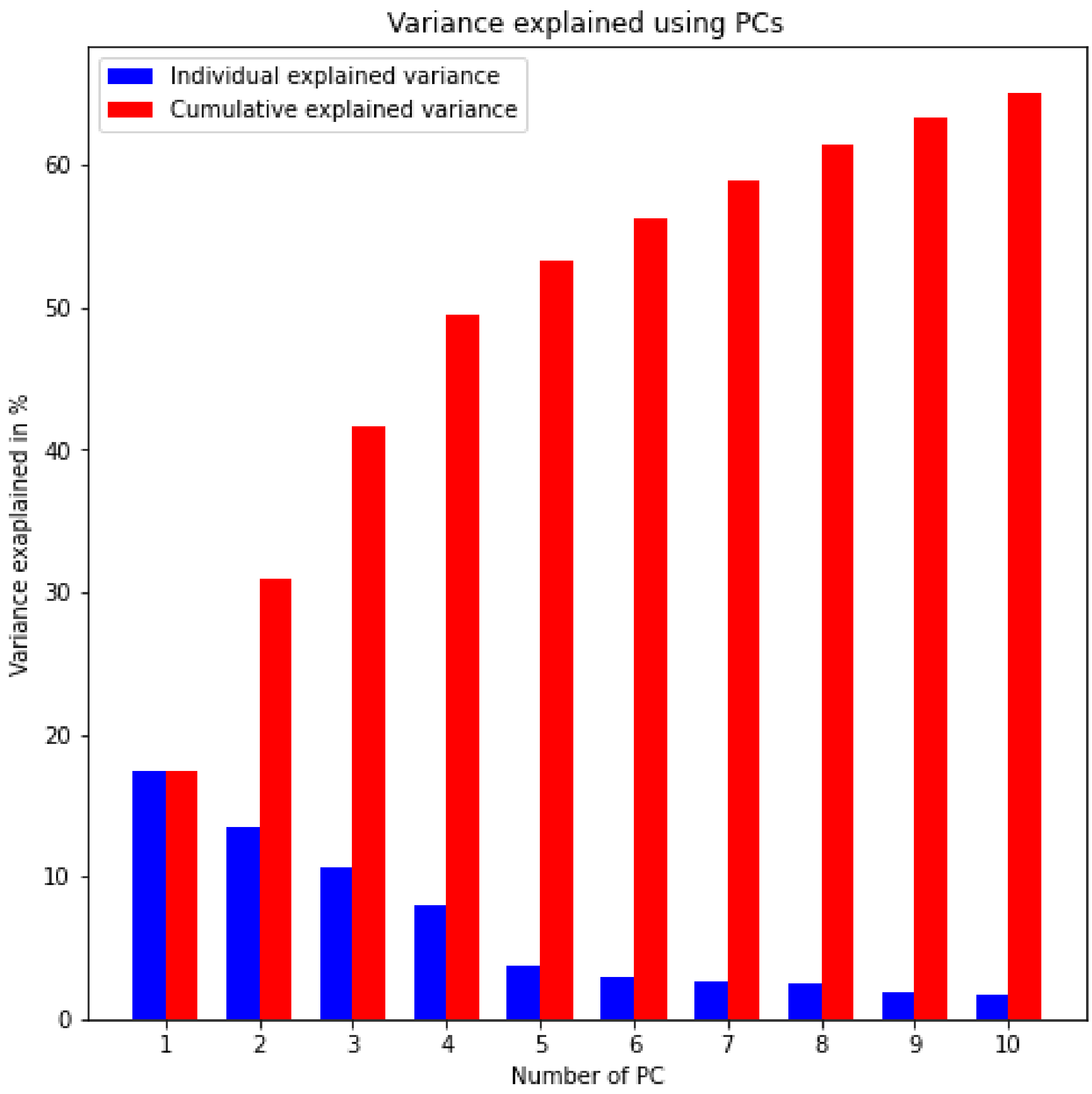
```
        var_tmp=var_tmp+eigval[i]
        n_PC.append(i)
        var_temp.append(eigval[i]/(eigval.sum())*100)
        var_explained.append(var_tmp/(eigval.sum())*100)

    fig, ax = plt.subplots(figsize=(8,8))

    ind = np.arange(10)
    width = 0.35         # the width of the bars
    p1 = ax.bar(ind, var_temp, width, color='b')
    p2 = ax.bar(ind + width, var_explained, width, color='r')

    ax.legend((p1[0], p2[0]), ('Individual explained variance', 'Cumulative␣
↪explained variance'))

    ax.set_title('Variance explained using PCs')
    ax.set_xticks(ind + width / 2)
    ax.set_xticklabels(('1', '2', '3', '4', '5', '6', '7', '8', '9', '10'))

    plt.xlabel('Number of PC')
    plt.ylabel('Variance exaplained in %')

    ax.autoscale_view()

    plt.show()
```

[17]:
```
image_id = 2
image = X_t[image_id]

#Compute PCA
X_2 = computePCA(2, X_train,image_id)
X_10 = computePCA(10, X_train,image_id)
X_50 = computePCA(50, X_train,image_id)

#Reshape in order to plot images
X2 = np.reshape(X_2, (dim,dim,3)).astype(int)
X10 = np.reshape(X_10, (dim,dim,3)).astype(int)
X50 = np.reshape(X_50, (dim,dim,3)).astype(int)

#Plot
showPCA(image, X2, X10, X50)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

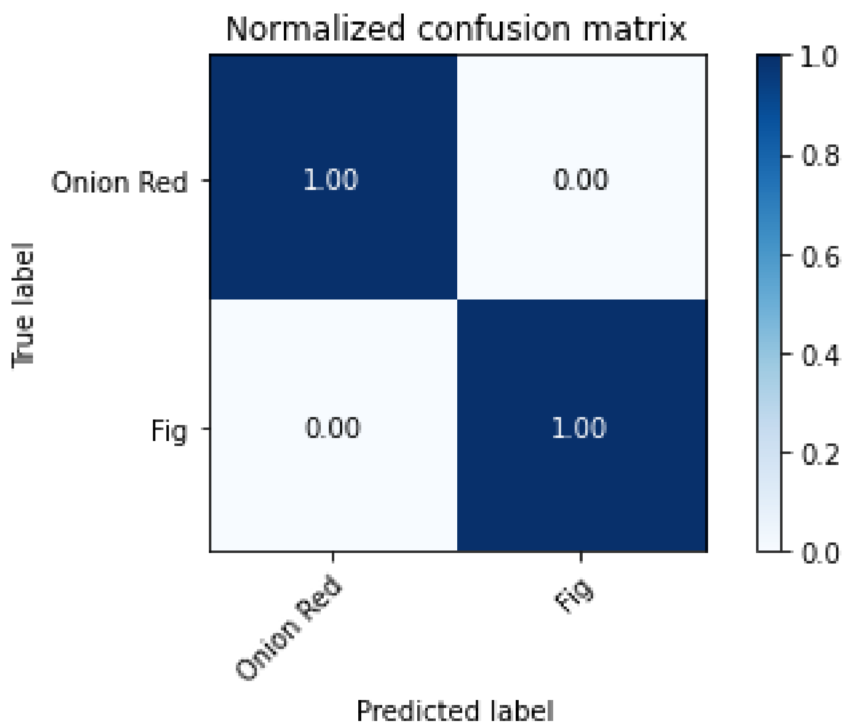


[18]:
```
showVariance(X_train)
```



[19]:
```python
# SVM without PCA
svm = SVC(gamma='auto', kernel='linear', probability=True)
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)

#Evaluation
precision = metrics.accuracy_score(y_pred, y_test) * 100
print("Accuracy with SVM: {0:.2f}%".format(precision))
cm , _ = plot_confusion_matrix(y_test, y_pred,classes=y_train, normalize=True,
 ↪title='Normalized confusion matrix')
plt.show()

# calculate the FPR and TPR for all thresholds of the classification
probs = svm.predict_proba(X_test)
probs = probs[:, 1]
svm_fpr, svm_tpr, thresholds = metrics.roc_curve(y_test, probs)
svm_auc = metrics.roc_auc_score(y_test, probs)
```

```
Accuracy with SVM: 100.00%
```

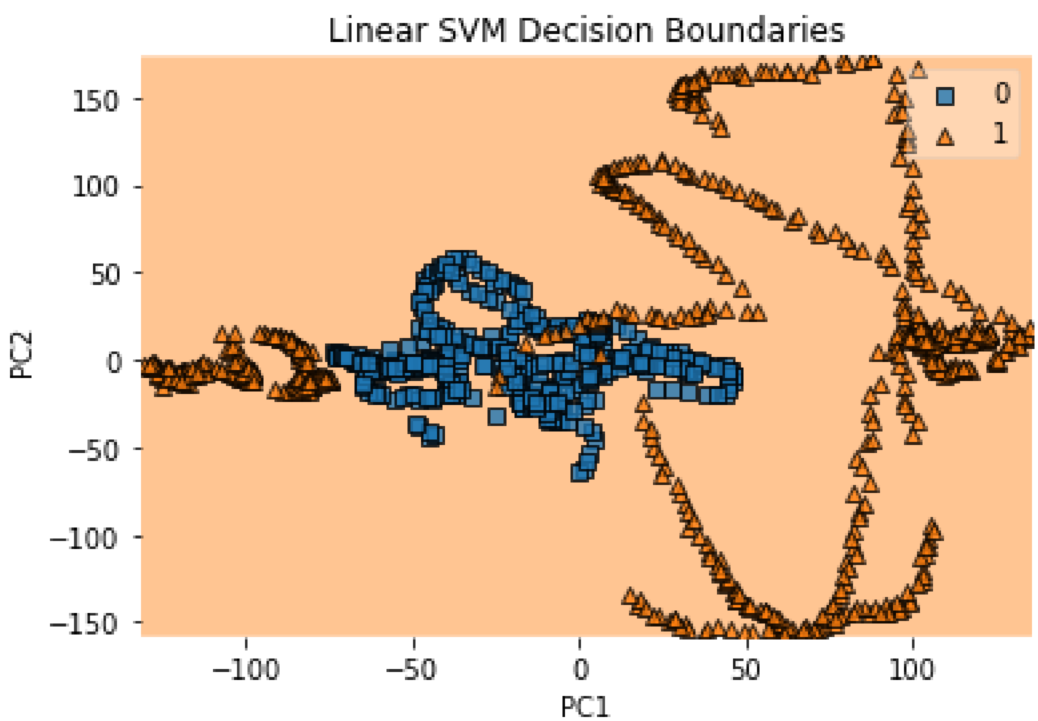


[20]:
```python
# SVM with PCA
pca = PCA(n_components=2)
X_train2D = pca.fit_transform(X_train)
```

```python
X_test2D = pca.fit_transform(X_test)

svm.fit(X_train2D, y_train)
test_predictions = svm.predict(X_test2D)
precision = metrics.accuracy_score(test_predictions, y_test) * 100
print("Accuracy with SVM considering only first 2PC: {0:.2f}%".
 ↪format(precision))

#Plotting decision boundaries
plot_decision_regions(X_train2D, y_train, clf=svm, legend=1)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('Linear SVM Decision Boundaries')
plt.show()
```

```
Accuracy with SVM considering only first 2PC: 60.94%
```

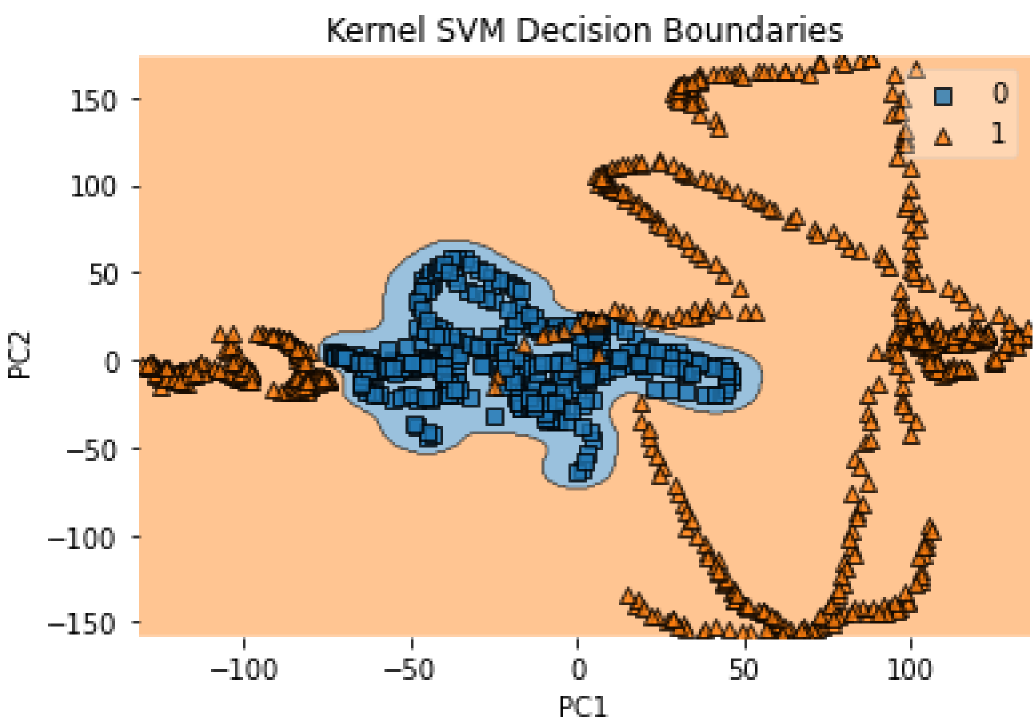
```
/usr/local/lib/python3.7/dist-packages/mlxtend/plotting/decision_regions.py:242:
UserWarning: No contour levels were found within the data range.
  antialiased=True)
/usr/local/lib/python3.7/dist-packages/mlxtend/plotting/decision_regions.py:244:
MatplotlibDeprecationWarning: Passing unsupported keyword arguments to axis()
will raise a TypeError in 3.3.
  ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())
```



[21]:
```
# SVM with kernel
svm_with_kernel = SVC(gamma=0.01, kernel='rbf', probability=True)
svm_with_kernel.fit(X_train2D, y_train)
y_pred = svm_with_kernel.predict(X_test2D)
precision = metrics.accuracy_score(y_pred, y_test) * 100
print("Accuracy with Not-Linear SVM considering only first 2PC: {0:.2f}%".
 ↪format(precision))

#Plotting decision boundaries
plot_decision_regions(X_train2D, y_train, clf=svm_with_kernel, legend=1)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('Kernel SVM Decision Boundaries')
plt.show()
```

```
Accuracy with Not-Linear SVM considering only first 2PC: 62.50%
```

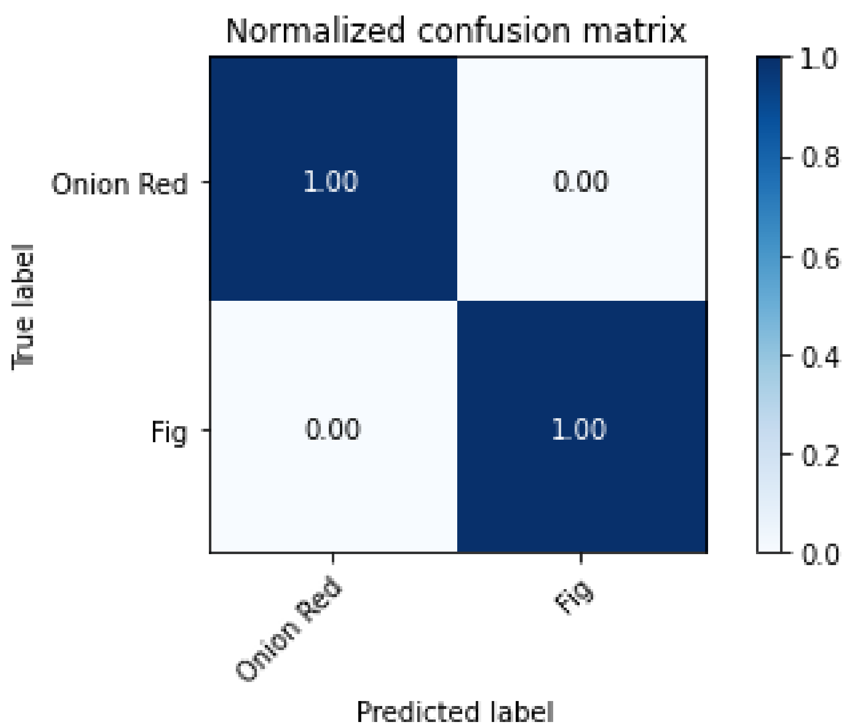
```
/usr/local/lib/python3.7/dist-packages/mlxtend/plotting/decision_regions.py:244:
MatplotlibDeprecationWarning: Passing unsupported keyword arguments to axis()
will raise a TypeError in 3.3.
  ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())
```



[22]:
```
# decision tree classifier
tree = DecisionTreeClassifier()
```

```
tree = tree.fit(X_train,y_train)
y_pred = tree.predict(X_test)

#Evaluation
precision = metrics.accuracy_score(y_pred, y_test) * 100
print("Accuracy with Decision Tree: {0:.2f}%".format(precision))
cm , _ = plot_confusion_matrix(y_test, y_pred, classes=y_train, normalize=True,␣
 ↪title='Normalized confusion matrix')
plt.show()

# calculate the FPR and TPR for all thresholds of the classification
probs = tree.predict_proba(X_test)
probs = probs[:, 1]
tree_fpr, tree_tpr, thresholds = metrics.roc_curve(y_test, probs)
tree_auc = metrics.roc_auc_score(y_test, probs)
```

```
Accuracy with Decision Tree: 100.00%
```



[24]:
```
# bigger dataset
fruits = ['Apple Red 1','Apricot','Avocado','Banana','Blueberry','Cherry␣
 ↪1','Cocos','Dates','Fig','Grape␣
 ↪White','Guava','Hazelnut','Kiwi','Lemon','Mango','Orange','Papaya','Peach','Pear','Pineappl
print(fruits)
```

```
['Apple Red 1', 'Apricot', 'Avocado', 'Banana', 'Blueberry', 'Cherry 1',
'Cocos', 'Dates', 'Fig', 'Grape White', 'Guava', 'Hazelnut', 'Kiwi', 'Lemon',
'Mango', 'Orange', 'Papaya', 'Peach', 'Pear', 'Pineapple', 'Plum',
'Pomegranate', 'Strawberry', 'Walnut', 'Watermelon']
```

[25]:

```python
X, y =  getDataset(fruits, 'Training', print_n=True, k_fold=False)
X_test, y_test = getDataset(fruits, 'Test', print_n=True, k_fold=False)
```

```
There are  492   TRAINING  images of  APPLE RED 1
There are  492   TRAINING  images of  APRICOT
There are  427   TRAINING  images of  AVOCADO
There are  490   TRAINING  images of  BANANA
There are  462   TRAINING  images of  BLUEBERRY
There are  492   TRAINING  images of  CHERRY 1
There are  490   TRAINING  images of  COCOS
There are  490   TRAINING  images of  DATES
There are  702   TRAINING  images of  FIG
There are  490   TRAINING  images of  GRAPE WHITE
There are  490   TRAINING  images of  GUAVA
There are  464   TRAINING  images of  HAZELNUT
There are  466   TRAINING  images of  KIWI
There are  492   TRAINING  images of  LEMON
There are  490   TRAINING  images of  MANGO
There are  479   TRAINING  images of  ORANGE
There are  492   TRAINING  images of  PAPAYA
There are  492   TRAINING  images of  PEACH
There are  492   TRAINING  images of  PEAR
There are  490   TRAINING  images of  PINEAPPLE
There are  447   TRAINING  images of  PLUM
There are  492   TRAINING  images of  POMEGRANATE
There are  492   TRAINING  images of  STRAWBERRY
There are  735   TRAINING  images of  WALNUT
There are  475   TRAINING  images of  WATERMELON
There are  164   TEST  images of  APPLE RED 1
There are  164   TEST  images of  APRICOT
There are  143   TEST  images of  AVOCADO
There are  166   TEST  images of  BANANA
There are  154   TEST  images of  BLUEBERRY
There are  164   TEST  images of  CHERRY 1
There are  166   TEST  images of  COCOS
There are  166   TEST  images of  DATES
There are  234   TEST  images of  FIG
There are  166   TEST  images of  GRAPE WHITE
There are  166   TEST  images of  GUAVA
There are  157   TEST  images of  HAZELNUT
There are  156   TEST  images of  KIWI
There are  164   TEST  images of  LEMON
There are  166   TEST  images of  MANGO
```

```
There are  160   TEST  images of  ORANGE
There are  164   TEST  images of  PAPAYA
There are  164   TEST  images of  PEACH
There are  164   TEST  images of  PEAR
There are  166   TEST  images of  PINEAPPLE
There are  151   TEST  images of  PLUM
There are  164   TEST  images of  POMEGRANATE
There are  164   TEST  images of  STRAWBERRY
There are  249   TEST  images of  WALNUT
There are  157   TEST  images of  WATERMELON
```

[26]:

```
#Scale Data Images
scaler = StandardScaler()
X_train = scaler.fit_transform([i.flatten() for i in X])
X_test = scaler.fit_transform([i.flatten() for i in X_test])
```

[27]:

```
#SVM
model = SVC(gamma='auto', kernel='linear')
model.fit(X_train, y)
y_pred = model.predict(X_test)
precision = metrics.accuracy_score(y_pred, y_test) * 100
print("Accuracy with SVM: {0:.2f}%".format(precision))
```

```
Accuracy with SVM: 98.98%
```