ML_prog4

July 30, 2020

1 ML LAB PROGRAM 4

2 Demonstrating PCA

2.1 Importing necessary libraries

[1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import image
import seaborn as sns; sns.set()
import glob
from sklearn.utils import shuffle
from PIL import Image
import os, sys
from mpl_toolkits.mplot3d import Axes3D
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.decomposition import PCA
```

2.2 Resizing all of the images in the dataset to 64 x 64

2.2.1 Data consists of 10000 images of cats and dogs that are of different sizes.

[2]:
```
paths = ["D:\\Education\\MSc\\Sem 3\\lab\\ML\\ML_prog4\\training_set\\cats\\",
         "D:\\Education\\MSc\\Sem 3\\lab\\ML\\ML_prog4\\training_set\\dogs\\",
         "D:\\Education\\MSc\\Sem 3\\lab\\ML\\ML_prog4\\test_set\\cats\\",
         "D:\\Education\\MSc\\Sem 3\\lab\\ML\\ML_prog4\\test_set\\dogs\\"]
def resize():
    for path in paths:
        dirs = os.listdir( path )

        for item in dirs:
            if os.path.isfile(path+item):
                im = Image.open(path+item)
                f, e = os.path.splitext(path+item)
                imResize = im.resize((32,32), Image.ANTIALIAS)
                imResize.save(f+'.jpg', 'JPEG', quality=90)

resize()
```

2.3 Preparing training data

2.3.1 Using glob to list files (We are going to read only half the data i.e 4000 images for train set)**

** Due to memory constraints

[3]:
```
flist_cat = glob.glob("training_set/cats/*.jpg")
flist_cat = flist_cat[:int(len(flist_cat)/2)]
print(len(flist_cat))
flist_dog = glob.glob("training_set/dogs/*.jpg")
flist_dog = flist_dog[:int(len(flist_dog)/2)]
print(len(flist_dog))
```

```
2000
2002
```

2.3.2 Read images (Using matlplotlib.image) into a numpy array

Image sizes are displayed

[4]:
```
cats = [image.imread(i) for i in flist_cat if image.imread(i).shape[0] == 32 and␣
 ↪image.imread(i).shape[1] == 32]
dogs = [image.imread(i) for i in flist_dog if image.imread(i).shape[0] == 32 and␣
 ↪image.imread(i).shape[1] == 32]
print(len(cats))
print(len(dogs))
train_set_x = np.array(cats+dogs)
print(type(train_set_x))
print(train_set_x.shape)
```

```
2000
2002
<class 'numpy.ndarray'>
(4002, 32, 32, 3)
```

2.3.3 Preparing a target array for the train set.

2.3.4 Finally we have train_set_x (train set data) and train_set_y (targets)

Size of arrays are displayed

[5]:
```
cats_y = np.ones([1,len(cats)])
dogs_y = np.zeros([1,len(dogs)])
print(cats_y.shape)
print(dogs_y.shape)
train_set_y = np.concatenate((cats_y,dogs_y),axis = 1).T
print(train_set_y.shape)
```

```
(1, 2000)
(1, 2002)
(4002, 1)
```

2.4 Preparing test set

2.4.1 Using glob to list files (We are reading about half the images i.e 500)**

** Due to memory constraints

[6]:
```
flist_cat_test = glob.glob("test_set/cats/*.jpg")
flist_cat_test = flist_cat_test[:int(len(flist_cat_test)/2)]
print(len(flist_cat_test))
flist_dog_test = glob.glob("test_set/dogs/*.jpg")
flist_dog_test = flist_dog_test[:int(len(flist_dog_test)/2)]
print(len(flist_dog_test))
```

```
505
506
```

2.4.2 Reading the images into numpy array

Size of images are displayed

[7]:
```
cats_test = [image.imread(i) for i in flist_cat_test if image.imread(i).shape[0]␣
 ↪== 32 and image.imread(i).shape[1] == 32]
dogs_test = [image.imread(i) for i in flist_dog_test if image.imread(i).shape[0]␣
 ↪== 32 and image.imread(i).shape[1] == 32]
print(len(cats_test))
print(len(dogs_test))
test_set_x = np.array(cats_test+dogs_test)
print(type(test_set_x))
print(test_set_x.shape)
```

```
505
506
<class 'numpy.ndarray'>
(1011, 32, 32, 3)
```

2.4.3 Preparing targets for test set

2.4.4 Finally we have test_set_x (test set data) and test_set_y (targets)

Size of array is displayed

[8]:
```
cats_y_test = np.ones([1,len(cats_test)])
dogs_y_test = np.zeros([1,len(dogs_test)])
print(cats_y_test.shape)
print(dogs_y_test.shape)
test_set_y = np.concatenate((cats_y_test,dogs_y_test),axis = 1).T
print(test_set_y.shape)
```

```
(1, 505)
(1, 506)
(1011, 1)
```

2.5 Shuffling the dataset using sklearn's shuffle ( )

2.5.1 The sizes after shuffling are displayed

[9]:
```
train_set_x,train_set_y = shuffle(train_set_x,train_set_y)
test_set_x,test_set_y = shuffle(test_set_x,test_set_y)
```

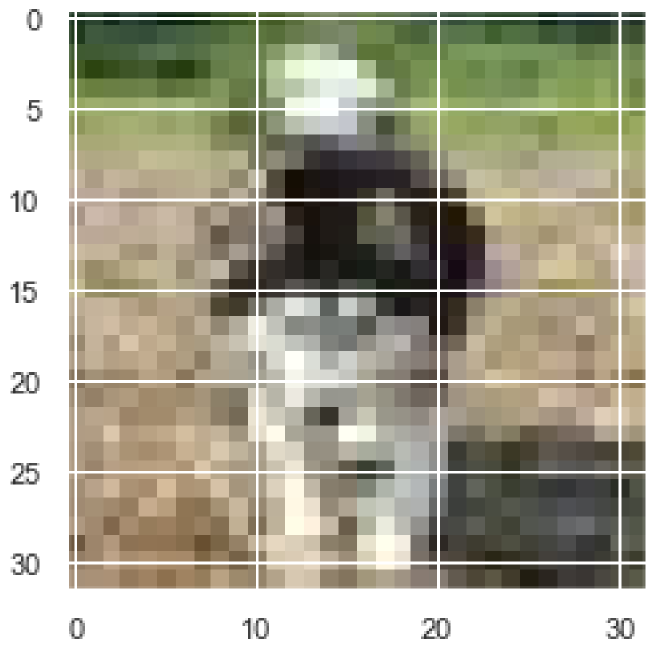
[10]:
```
print(train_set_x.shape)
print(train_set_y.shape)
print(test_set_x.shape)
print(test_set_y.shape)
```

```
(4002, 32, 32, 3)
(4002, 1)
(1011, 32, 32, 3)
(1011, 1)
```

2.6 Modify index (within bounds [0 , 4000] to view train data

[11]:
```
index = 2355
plt.imshow(train_set_x[index])
print("y = ",train_set_y[index])
```

```
y =  [0.]
```



2.6.1 Normalizing is not necesssary as all values are between 0 and 255

[37]:
```
#train_set_x_norm = (train_set_x - train_set_x.mean(axis=(0,1,2),␣
↪keepdims=True)) / train_set_x.std(axis=(0,1,2), keepdims=True)
#test_set_x_norm = (test_set_x - test_set_x.mean(axis=(0,1,2), keepdims=True)) /␣
↪test_set_x.std(axis=(0,1,2), keepdims=True)
```

2.7 Making a single vector out of a (64,64,3) image.

2.7.1 Resultant sizes are displayed

PCA cannot handle more than 2 dimensional arrays.

[13]:
```
train_vector = train_set_x.reshape(train_set_x.shape[0],-1)
test_vector = test_set_x.reshape(test_set_x.shape[0],-1)
print(train_vector.shape)
print(test_vector.shape)
```

```
(4002, 3072)
(1011, 3072)
```

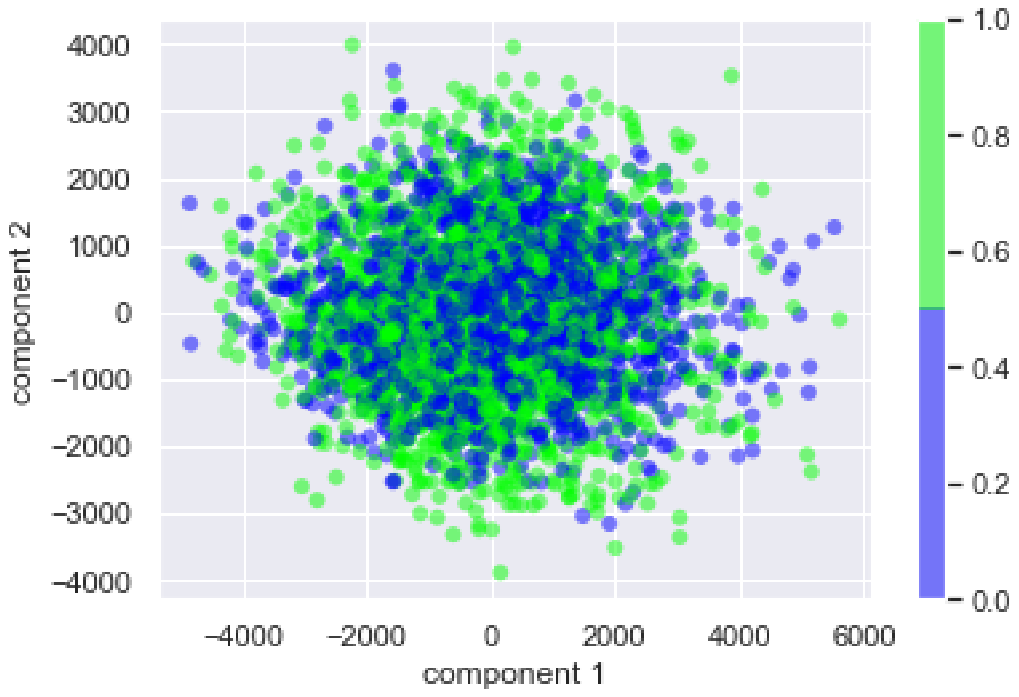
2.8 PCA for visualization of data

2.8.1 Using PCA ( ) to reduce dimension from 3072 to 2 (2D plot)

[38]:
```
pca = PCA(2)
projected = pca.fit_transform(train_vector)
print(train_vector.shape)
print(projected.shape)
print("%0.2f"%(np.cumsum(pca.explained_variance_ratio_)[-1]*100),"% of variance␣
 ↪is preserved")
```

```
(4002, 3072)
(4002, 2)
31.22 % of variance is preserved
```

[15]:
```
plt.scatter(projected[:, 0], projected[:, 1],
            c=train_set_y, edgecolor='none', alpha=0.5,
            cmap=plt.cm.get_cmap('brg', 2))
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.colorbar();
```

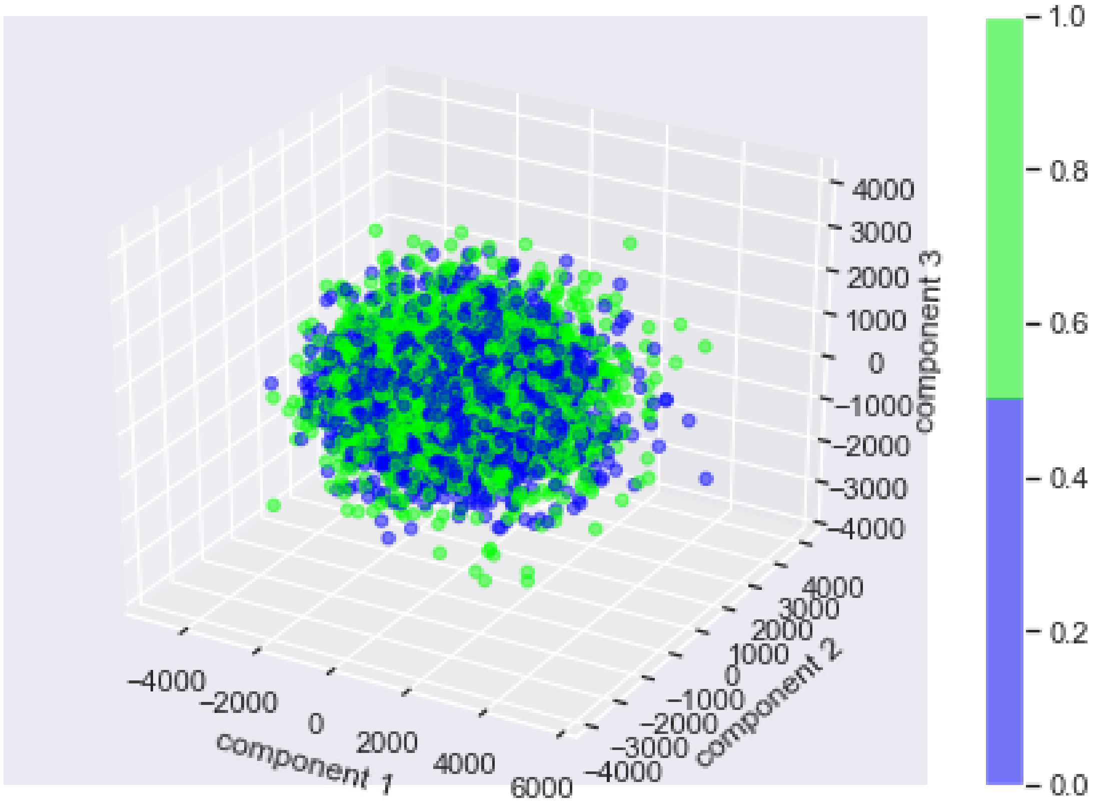


2.8.2 Using PCA ( ) to reduce dimesnionality from 3072 to 3 (3D plot)

[39]:
```
pca = PCA(3)
projected = pca.fit_transform(train_vector)
print(train_set_x.shape)
print(projected.shape)
print("%0.2f"%(np.cumsum(pca.explained_variance_ratio_)[-1]*100),"% of variance␣
 ↪is preserved")
```

```
(4002, 32, 32, 3)
(4002, 3)
39.04 % of variance is preserved
```

[17]:
```
ax = Axes3D(plt.figure())
p = ax.scatter(projected[:, 0], projected[:, 1],projected[:,2],
            c=train_set_y, alpha=0.5,
            cmap=plt.cm.get_cmap('brg', 2))
plt.xlabel('component 1')
plt.ylabel('component 2')
ax.set_zlabel('component 3')
plt.colorbar(p);
```



2.9 PCA ( ) to reduce dimensionality using number of components

```
[18]: pca = PCA(10)  # project from 64 to 2 dimensions
      projected = pca.fit_transform(train_vector)
      print(train_set_x.shape)
      print(projected.shape)
      print("%0.2f"%(np.cumsum(pca.explained_variance_ratio_)[-1]*100),"% of variance␣
       ↪is preserved")
```

```
(4002, 32, 32, 3)
(4002, 10)
59.79 % of variance is preserved
```

2.10 PCA ( ) to reduce dimensionality using percentage of explained variance

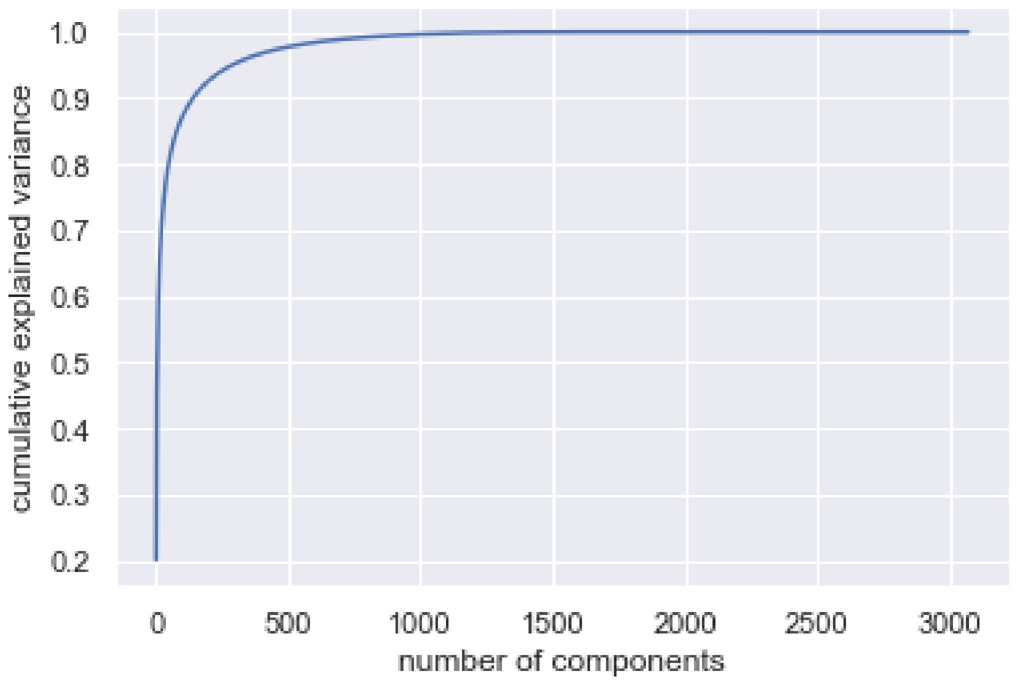
```
[19]: pca = PCA(0.9)
      projected = pca.fit_transform(train_vector)
      print(train_set_x.shape)
      print(projected.shape)
      print("%0.2f"%(np.cumsum(pca.explained_variance_ratio_)[-1]*100),"% of variance␣
       ↪is preserved")
```

```
(4002, 32, 32, 3)
(4002, 141)
90.04 % of variance is preserved
```

2.11 Visualizing the number of dimensions vs the preserved variance plot

2.11.1 We can see that for about 145 dimensions we get 90% variance

```
[20]: pca = PCA().fit(train_vector)
      plt.plot(np.cumsum(pca.explained_variance_ratio_))
      plt.xlabel('number of components')
      plt.ylabel('cumulative explained variance');
```



[21]:
```
cumsums = np.cumsum(pca.explained_variance_ratio_)
print(cumsums[144]*100)
```

```
90.2622239929583
```

2.12 Training a logistic regression model on original data set

2.12.1 Takes about 10 minutes to train

2.12.2 Accuracy is reported at the end

[22]:
```
model = LogisticRegression(solver='liblinear', random_state=0).
 ↪fit(train_vector,train_set_y)
```

```
D:\anaconda3\lib\site-packages\sklearn\utils\validation.py:73:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(**kwargs)
```

[23]:
```
print("Test accuracy: %0.2f "%(model.score(test_vector,test_set_y)*100),"%")
print("Train accuracy: %0.2f "%(model.score(train_vector,train_set_y)*100),"%")
```

```
Test accuracy: 56.97  %
Train accuracy: 100.00  %
```

2.13 Training logistic regression model on data after PCA ( ) is apllied

2.13.1 Using PCA ( ) to reduce dimensionality

[34]:
```
pca = PCA(700)  # project from 64 to 2 dimensions
projected_x = pca.fit_transform(train_vector)
projected_y = pca.fit_transform(test_vector)
#recon = pca.inverse_transform(projected)
print(train_vector.shape)
print(projected_x.shape)
print("%0.2f"%(np.cumsum(pca.explained_variance_ratio_)[-1]*100),"% of variance␣
 ↪is preserved")
```

```
(4002, 3072)
(4002, 700)
99.76 % of variance is preserved
```

2.13.2 Model takes a few seconds to train (1500% decrease in time taken or almost 150 times faster)

2.13.3 Accuracy is reported at the end

[35]:
```
model2 = LogisticRegression(solver='liblinear', random_state=0).
 ↪fit(projected_x,train_set_y)
```

```
D:\anaconda3\lib\site-packages\sklearn\utils\validation.py:73:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(**kwargs)
```

[36]:
```
print("Test accuracy: %0.2f "%(model2.score(projected_y,test_set_y)*100),"%")
print("Train accuracy: %0.2f "%(model2.score(projected_x,train_set_y)*100),"%")
```

```
Test accuracy: 53.12  %
Train accuracy: 71.79  %
```

2.13.4 Reducing training time by 150 times the original for a 4% drop in accuracy. With enough data this could be rectified

End of Notebook