MNIST_PCA

May 31, 2018

In [19]:
```
% matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix, accuracy_score
#from sklearn.datasets import load_digits
from sklearn.linear_model import LogisticRegression
from sklearn import svm
import seaborn as sns

import matplotlib.image as mpimg
from skimage.io import imread, imshow
```

In [20]:
```
#Inlezen van de dataset
df_train = pd.read_csv('train.csv')
df_test = pd.read_csv('test.csv')
```

In [21]:
```
#Opsplitsen in features en targets

y_train = df_train.label.values
y_test = df_test.label.values

X_train = df_train.drop("label",axis=1).values
X_test  = df_test.drop("label",axis=1).values
```

In [6]:
```
image_index = 8
print('label van het cijfer: ',y_train[image_index])
plt.imshow(X_train[image_index].reshape((28, 28)),cmap = 'gray')
plt.axis('off')
```

```
label van het cijfer:  5
```

Out[6]:
```
(-0.5, 27.5, 27.5, -0.5)
```

In [9]:
```
# Principle Component Analysis

number_of_components = 40

from sklearn.decomposition import PCA

#Train het PCA algoritme op de training data
pca_model = PCA(n_components=number_of_components, svd_solver='full')
pca_model.fit(X_train)

#Reduceer het aantal dimensies van zowel de trainig set als de test set
X_train_pca = pca_model.transform(X_train)
X_test_pca = pca_model.transform(X_test)
```

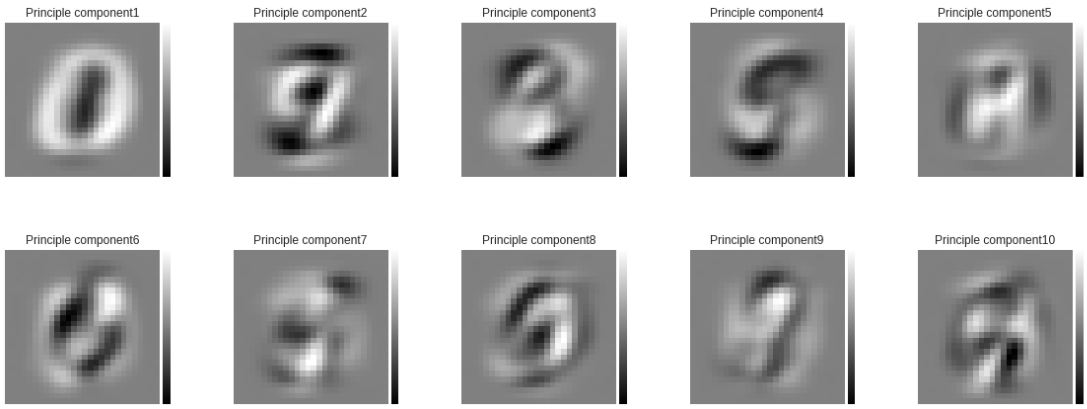
In [10]:
```
# Visualisatie van de eerste 10 principle components

 PCA_components = pca_model.components_

 fig = plt.figure(figsize=(16, 9))
 for i in range(0,10):
```

```
        ax = fig.add_subplot(2, 5, i+1)
        imshow(PCA_components[i].reshape((28,28)),cmap='gray')
        ax.xaxis.set_visible(False)
        ax.yaxis.set_visible(False)
        plt.axis('off')
        plt.legend('off')

        ax.set_title('Principle component'+str(i+1))
        # x and y axis should be equal length
        x0,x1 = ax.get_xlim()
        y0,y1 = ax.get_ylim()
    plt.show()
```



In [11]: *#Geef de PCA scores van het gekozen getal*

```
print(X_train_pca[image_index,:])
```

```
[ 354.77055124  143.18785282 -381.81672672 -620.98948171  310.27798688
 -489.44089344 -533.56768182 -232.51585886 -366.40133669   64.6972105
  -35.69910938  134.18047953 -102.48101831  586.58141887  221.9634094
 -212.51414373 -131.20597816 -166.19094881 -149.4476211   290.90395739
   70.93182371  -27.67287027  -72.74812974   80.20823245 -144.27449129
  237.1693674   -83.44465842  120.33772477   -0.92671744 -109.60021222
 -160.50045372  101.39860199    7.92286194 -336.53555756 -208.17877704
  164.90558562   98.66395481  -60.81260535   23.4675592    42.27910839]
```

In [12]: *# Geeft weer hoeveel van de variantie door elke principle component wordt verklaard*

```
print(pca_model.explained_variance_ratio_)
```

```
[ 0.09622774  0.07169274  0.06162532  0.05412628  0.04859137  0.04297465
  0.03292357  0.02892797  0.02755892  0.02348328  0.02107819  0.02052552
```

```
  0.01707707  0.01688598  0.01583215  0.01481815  0.01322416  0.0129219
  0.0118972   0.01157611  0.01076297  0.01013138  0.00966695  0.00918594
  0.00886457  0.00835795  0.00812716  0.00789175  0.0074726   0.00690113
  0.00658793  0.00645775  0.00606518  0.00595052  0.00560355  0.00542467
  0.00507373  0.0048704   0.00479924  0.0046529 ]
```

In [13]:
```
# Geef weer hoeveel van de variantie in totaal wordt verklaard door het model

print(np.sum(pca_model.explained_variance_ratio_))
```

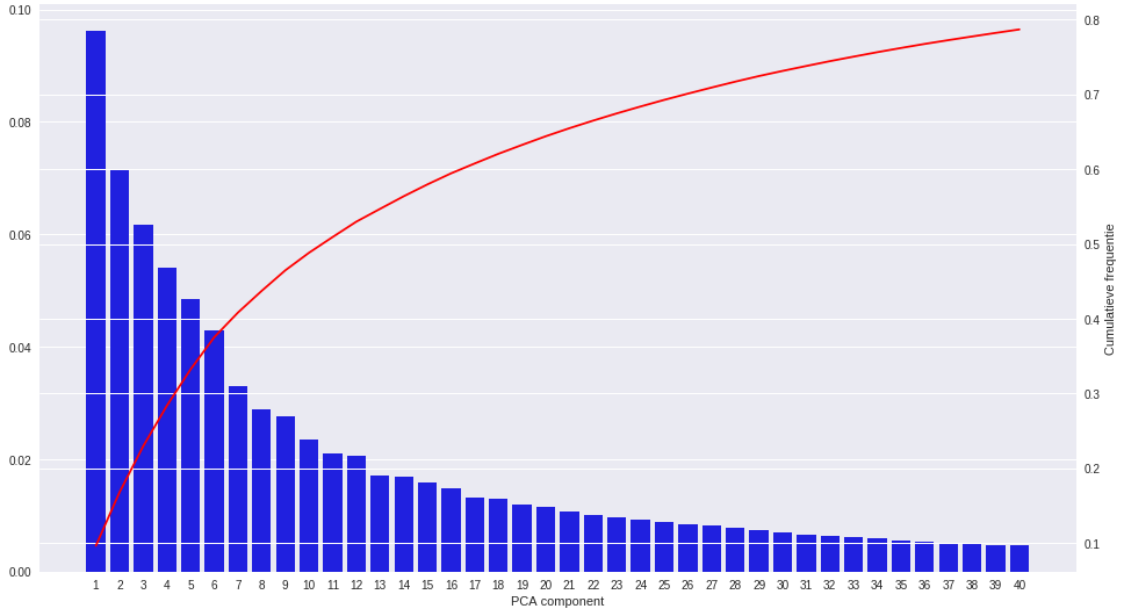
```
0.786816536595
```

In [14]:
```
PCAnumber = np.arange(1,number_of_components+1)
print(PCAnumber)
PCA_explained_cumulative = np.cumsum(pca_model.explained_variance_ratio_)
print(PCA_explained_cumulative)

fig = plt.figure(figsize=(16, 9))
ax = sns.barplot(PCAnumber,pca_model.explained_variance_ratio_,color='blue')
plt.xlabel('PCA component')
ax2=ax.twinx() # dubbele y-as
ax.yaxis.set_label_position('left')
ax2.yaxis.set_label_position('right')
ax2.set_ylabel('Cumulatieve frequentie')
plt.plot(PCA_explained_cumulative, c='red')
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40]
[ 0.09622774  0.16792049  0.2295458   0.28367208  0.33226345  0.3752381
  0.40816167  0.43708964  0.46464856  0.48813184  0.50921003  0.52973555
  0.54681262  0.5636986   0.57953075  0.5943489   0.60757306  0.62049496
  0.63239216  0.64396826  0.65473123  0.66486261  0.67452957  0.68371551
  0.69258007  0.70093802  0.70906518  0.71695693  0.72442953  0.73133066
  0.73791859  0.74437634  0.75044152  0.75639205  0.7619956   0.76742027
  0.772494    0.7773644   0.78216364  0.78681654]
```

Out[14]:
```
[<matplotlib.lines.Line2D at 0x7f231198e860>]
```



In [15]:
```
# Reconstrueer de gekozen afbeelding

image_reconstructed = pca_model.inverse_transform(X_train_pca[image_index,:])
print(image_reconstructed.shape)

plt.imshow(image_reconstructed.reshape((28, 28)),cmap = 'gray')
plt.axis('off')
```

```
(784,)
```

Out[15]:
```
(-0.5, 27.5, 27.5, -0.5)
```

1 Train een logistic regression classifier op gereduceerde training set

```
In [24]: lregmodel = LogisticRegression(multi_class='multinomial', solver='lbfgs')
         lregmodel.fit(X_train_pca,y_train)

Out[24]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='multinomial',
                   n_jobs=1, penalty='l2', random_state=None, solver='lbfgs',
                   tol=0.0001, verbose=0, warm_start=False)

In [23]: testindex = 0
         print('Werkelijk cijfer: ',y_test[testindex])
         prediction = lregmodel.predict(X_test_pca[testindex].reshape(1,-1))
         print('Herkend cijfer: ',prediction)

Werkelijk cijfer:  3
Herkend cijfer:  [3]
```

In [18]:
```
# Evaluatie van de classifier

y_predicted = lregmodel.predict(X_test_pca)

# Confusion matrix

print(confusion_matrix(y_test, y_predicted))

# Classification report

print(classification_report(y_test, y_predicted))

print(accuracy_score(y_test, y_predicted))
```

```
[[1150    0    5    5    0    9   10    2   13    1]
 [   0 1321    4    3    0    4    1    3   15    1]
 [  10   11 1013   12   19   11   24   15   36    6]
 [   6   10   32 1097    0   54    9   11   23   16]
 [   3    7   11    1 1030    2   18    0    9   59]
 [  15   15   12   39   14  898   24    5   38   16]
 [  15    3   11    0   17   14 1094    2   11    0]
 [   6    7   19    4   11    5    0 1165    3   48]
 [  11   36   12   29    6   45    8    6 1005   16]
 [   9    8    9   14   48   15    1   52   13 1044]]
             precision    recall  f1-score   support

          0       0.94      0.96      0.95      1195
          1       0.93      0.98      0.95      1352
          2       0.90      0.88      0.89      1157
          3       0.91      0.87      0.89      1258
          4       0.90      0.90      0.90      1140
          5       0.85      0.83      0.84      1076
          6       0.92      0.94      0.93      1167
          7       0.92      0.92      0.92      1268
          8       0.86      0.86      0.86      1174
          9       0.86      0.86      0.86      1213

avg / total       0.90      0.90      0.90     12000

0.901416666667
```

2 Train een SVM op gereduceerde training set

In [17]:
```
SVMmodel = svm.SVC(kernel='rbf',C=1,gamma=1)

SVMmodel.fit(X_train_pca, y_train)
```

```
Out[17]: SVC(C=1, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape=None, degree=3, gamma=1, kernel='rbf',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)
```

```
In [18]: # Evaluatie van de SVM classifier

         y_predicted_SVM = SVMmodel.predict(X_test_pca)
         # Confusion matrix

         print(confusion_matrix(y_test, y_predicted))

         # Classification report

         print(classification_report(y_test, y_predicted))

         print(accuracy_score(y_test, y_predicted))
```

```
[[1068    0    7   10    1   77   21    6    5    0]
 [   0 1306   11    4    0    5    6    1   18    1]
 [  32   16  914   29   22    9   65   21   46    3]
 [  14   16   41  952    2   92   17   17   86   21]
 [   2    8    6    0  906    7   42    6   10  153]
 [  34   51   12   81   29  718   40   24   55   32]
 [  26   23   54    1   26   26 1000    2    9    0]
 [  11   26   14    2   17   11    0 1102   24   61]
 [   5   71   42   84   10   64    8    2  833   55]
 [  16    6   22   16  191   25    5  110   23  799]]
              precision    recall  f1-score   support

           0       0.88      0.89      0.89      1195
           1       0.86      0.97      0.91      1352
           2       0.81      0.79      0.80      1157
           3       0.81      0.76      0.78      1258
           4       0.75      0.79      0.77      1140
           5       0.69      0.67      0.68      1076
           6       0.83      0.86      0.84      1167
           7       0.85      0.87      0.86      1268
           8       0.75      0.71      0.73      1174
           9       0.71      0.66      0.68      1213

avg / total       0.80      0.80      0.80     12000

0.799833333333
```

```
In [ ]:
```