Q3

June 16, 2020

1 Question 3. Image Classification

Dimensionality reduction, feature extraction and selection are crucial parts of high multidimensional data analysis. Consider a set of K training samples $X^{(k)} \in R^{I_1 x I_2 x .... x I_N}$, (k=1,2, ... , K) corresponding to C categories/classes, and a set of test data $X^t \in R^{I_1 x I_2 x .... x I_N}$, (t=1,2, ... , T). The challenge is to find appropriate labels for the test data. The classification algorithm can be generally performed in the following steps:

1. Find a set of basis matrices and the corresponding features from the training data $X^{(k)}$. The relation of a sample $X^{(k)}$ and basis factors can be expressed as:

$$X^{(k)} \approx G^{(k)} x_1 A^{(1)} x_2 A^{(2)} ... x_N A^{(N)} (k = 1, 2, ...K) \tag{1}$$

Where the core tensor $G^{(k)} \in R^{J_1 x J_2 x ... x J_N}$ representing features of a much lower dimension than the training data $X^{(k)}$. In other words, the core tensor $G^{(k)}$ consists of features of $X^{(k)}$ in subspace $A^{(n)}$.

2. Perform feature extraction for the test samples $X^{(t)}$ using the basis factors found for the training data (using a projected filter).

$$X^{(t)} x_1 A^{(1)T} ... x_N A^{(N)T} \tag{2}$$

3. Perform classification by comparing the test features with the training features.

You are given 28 training images, train1.jpg through train28.jpg. The first 14 images correspond to cats, and the remaining images correspond to birds. There are two classes: cats and birds. The labels for the images can be found in the file train_lab.mat. Your job will be to classify 12 new images, Test1.jpg through Test12.jpg. Use the training features to train a random forest with 100 trees. Note that you will need to vectorize the training features.

[1]:
```python
from scipy.io import loadmat
import matplotlib.pyplot as plt
import numpy as np
import tensorly as tl
from tensorly.tenalg import khatri_rao
from tensorly.decomposition import tucker
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import os
import re
import cv2
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```

```
<IPython.core.display.HTML object>
```

[2]:
```
# Helper functions for properly sorting the train and test images by name
def atoi(text):
    return int(text) if text.isdigit() else text

def natural_keys(text):
    '''
    alist.sort(key=natural_keys) sorts in human order
    http://nedbatchelder.com/blog/200712/human_sorting.html
    (See Toothy's implementation in the comments)
    '''
    return [ atoi(c) for c in re.split(r'(\d+)', text) ]
```

[3]:
```
#list of image paths for train and test images
train = [os.path.join('train', x) for x in sorted(os.listdir('train'),␣
 ↪key=natural_keys)]
test = [os.path.join('test', x) for x in sorted(os.listdir('test'),␣
 ↪key=natural_keys)]
```

[4]:
```
#load train and test labels
train_labs = loadmat('train_lab.mat')['train']
test_labs = loadmat('test_lab.mat')['test']
```

1.0.1 Part 1. Read and convert all images into gray scale. Form a third-order tensor using the training data and apply Tucker decomposition with $R_1 = 10; R_2 = 10; R_3 = 28$. Predict the labels for the images on the test set. Report the classification error.

[5]:
```
#load and convert to grayscale
grays_train = [cv2.cvtColor(plt.imread(x), cv2.COLOR_RGB2GRAY) for x in train]
grays_test = [cv2.cvtColor(plt.imread(x), cv2.COLOR_RGB2GRAY) for x in test]
```

[6]:
```
#Form a third order tensor of the gray images
X_train = np.dstack(grays_train)
X_test = np.dstack(grays_test)
```

[7]:
```
#apply Tucker Decmoposition
core, factors = tucker(tl.tensor(X_train, dtype=tl.float32), ranks=[10,10,28])
```

```
[8]: #abosrb core into the 3rd dimension factor since only the first 2 factors are␣
      ↪needed
     core=tl.tenalg.mode_dot(core, factors[-1], mode=2)
     #vectorize the core tensor
     core = core.reshape(-1,28).T
```

```
[9]: #create and train the RF classifier
     clf = RandomForestClassifier()
     clf.fit(core, train_labs.ravel())
```

```
[9]: RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                            criterion='gini', max_depth=None, max_features='auto',
                            max_leaf_nodes=None, max_samples=None,
                            min_impurity_decrease=0.0, min_impurity_split=None,
                            min_samples_leaf=1, min_samples_split=2,
                            min_weight_fraction_leaf=0.0, n_estimators=100,
                            n_jobs=None, oob_score=False, random_state=None,
                            verbose=0, warm_start=False)
```

```
[10]: #transform test tensor
      ttc = tl.tenalg.mode_dot(tl.tenalg.mode_dot(X_test, factors[0].T, mode=0),␣
       ↪factors[1].T, mode=1)
      #vectorize the tensor
      ttc=ttc.reshape(-1,12).T
```

```
[11]: #predict on test set
      preds = clf.predict(ttc)
```

```
[12]: #calculate accuracy on test set
      acc = accuracy_score(test_labs.ravel(), preds)
```

```
[13]: print('Accuracy on test set: {}'.format(acc))
      print('{} out of {} correctly predicted'.format(len(preds) - sum(test_labs.
       ↪ravel() - preds), len(preds)))
```

```
Accuracy on test set: 0.9166666666666666
11 out of 12 correctly predicted
```

1.0.2 Part 2. Read all images in RGB format. Form a fourth-order tensor using the training data and apply Tucker decomposition with $R_1 = 10; R_2 = 10; R_3 = 3; R_4 = 28$. Predict the labels for the images on the test set. Report the classification error.

[14]:
```
#Create 4th order tensor with the RGB images
rgb_train = [plt.imread(x) for x in train]
rgb_test = [plt.imread(x) for x in test]
```

[15]:
```
X_train = np.stack(rgb_train, axis=3)
X_test = np.stack(rgb_test, axis=3)
```

[16]:
```
#apply Tucker Decmoposition
core, factors = tucker(tl.tensor(X_train, dtype=tl.float32), ranks=[10,10,3,28])
```

[17]:
```
#abosrb core into the 4th dimension factor since only the first 3 factors are␣
↪needed
core=tl.tenalg.mode_dot(core, factors[-1], mode=3)
#vectorize the core tensor
core = core.reshape(-1, 28).T
```

[18]:
```
#create and train the RF classifier
clf = RandomForestClassifier()
clf.fit(core, train_labs.ravel())
```

[18]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

[19]:
```
#transform test tensor
ttc = tl.tenalg.mode_dot(tl.tenalg.mode_dot(tl.tenalg.mode_dot(X_test,␣
↪factors[0].T, mode=0), factors[1].T, mode=1), factors[2].T, mode=2)
#vectorize the tensor
ttc=ttc.reshape(-1,12).T
```

[20]:
```
#predict on test set
preds = clf.predict(ttc)
```

[21]:
```
#calculate accuracy on test set
acc = accuracy_score(test_labs.ravel(), preds)
```

```
[22]: print('Accuracy on test set: {}'.format(acc))
      print('{} out of {} correctly predicted'.format(len(preds) - sum(test_labs.
       ↪ravel() - preds), len(preds)))
```

```
Accuracy on test set: 0.9166666666666666
11 out of 12 correctly predicted
```