Demonstrating Image Classification using Random Forest Classifier Algorithm

Azhar Chowdhury
Statistical Programmer|Data Scientist[SAS, Python, R]

The following meta summary is helpful to follow the problem at hand and it's solution.

Dataset The MNIST dataset

Domain Image Processing

ML Algorithm Random Forest Classifier

Components Dimentionality Reduction Techniques, Dimentionality Reduction of Features.

Problem Description Apply a Random Forest classification algorithm to MNIST dataset Perform dimensionality reduction of features using PCA and compare classification on the reduced dataset to that of original one Apply dimensionality reduction techniques: t-SNE and LLE

[1]:
```
import pandas as pd
data = pd.read_csv('C://Users//azhar//Downloads//mnist_dataset.csv')
df = pd.DataFrame(data)
df.head(2)
```

[1]:

	Unnamed: 0	label	1x1	1x2	1x3	1x4	1x5	1x6	1x7	1x8	...	28x19	\
0	0	2	0	0	0	0	0	0	0	0	...	0	
1	1	5	0	0	0	0	0	0	0	0	...	0	

	28x20	28x21	28x22	28x23	28x24	28x25	28x26	28x27	28x28
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0

[2 rows x 786 columns]

[2]:
```
# Define X, y, training and testing portions - both target and response␣
↪variables of the dataset
X = df.iloc[:, :-1 ].values
y = df.iloc[:, -1].values
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```

[18]:
```
print("y:", y_train.shape, "\nX:", X_train.shape)
```

```
y: (60000,)
X: (60000, 785)
```

[4]:
```
%%time
from sklearn.ensemble import RandomForestClassifier # time on Entire set
rfc_T = RandomForestClassifier(max_depth=2, random_state=43) # rft_T = entire␣
↪dataset
rfc_T.fit(X, y)
```

```
CPU times: total: 1.31 s
Wall time: 1.93 s
```

[4]:
```
RandomForestClassifier(max_depth=2, random_state=43)
```

[5]:
```
%%time
from sklearn.ensemble import RandomForestClassifier # time on test set
rfc_t = RandomForestClassifier(max_depth=2, random_state=43)  # rft_t = test␣
↪dataset
rfc_t.fit(X_test, y_test)
```

```
CPU times: total: 141 ms
Wall time: 140 ms
```

[5]:
```
RandomForestClassifier(max_depth=2, random_state=43)
```

[6]:
```
# Reduction of dimentionality using PCA
import numpy as np
from sklearn.decomposition import PCA
pca = PCA()

pca.fit_transform(X)
evr = np.cumsum(pca.explained_variance_ratio_)
rDim = np.argmax(evr >= 0.95) +1
print(rDim.dtype) # [out:  int64] it is an integer and hence can be a parameter␣
↪for n_components=
#X_red = pca.fit_transform(X)
```

```
int64
```

[7]:
```
# Scaling the freatures at the same scale
import numpy as np
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
X_train_scaled = ss.fit_transform(X_train)
X_test_scaled = ss.transform(X_test)
y_train = np.array(y_train)

n = X_train.shape[1]  # n is the number of features before reduction.
```

[8]:
```
%%time
pca=PCA(n_components=n)
X_rT = pca.fit_transform(X_train)  # PCA on the training set.
```

```
evr = np.argmax(pca.explained_variance_ratio_ >= 0.95) + 1

print(evr)
```

```
1
CPU times: total: 19 s
Wall time: 2.72 s
```

[9]:
```
pca_v = PCA(n_components=evr)
pca_v.fit(X_train_scaled)
```

[9]:
```
PCA(n_components=1)
```

[10]:
```
%%time
pca=PCA(n_components=evr)
X_rDim = pca.fit_transform(X_test) # PCA on the test set.
```

```
CPU times: total: 859 ms
Wall time: 167 ms
```

The training sounds faster although we run the reduced feature classifier on the entire set of 70000 observations. On the Test set the Random Forest Classifier took: CPU times: total: 297 ms. Wall time: 309 ms. However, on applying PCA to reduce the features the same classifier took total: 78.1 ms, Wall time: 216 ms. It is evident that the classifier on the reduced feature runs much faster (78.1 ms on the reduced-feature test set, compared to 216 ms on the all-feature test set).

[11]:
```
import time
N=10000
import pandas as pd
#time_start = time.time()

features = ['pixel' + str(i) for i in range(X.shape[1])]
df = pd.DataFrame(X, columns=features)
df_tSNE = df.loc[:N, :].copy()
```

[12]:
```
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=300)
    # we're using this value from PCA's explained_variance_ratio_ value
tsne_res = tsne.fit_transform(df_tSNE[features].values)
```

```
[t-SNE] Computing 121 nearest neighbors...
[t-SNE] Indexed 10001 samples in 0.030s...
[t-SNE] Computed neighbors for 10001 samples in 2.341s...
[t-SNE] Computed conditional probabilities for sample 1000 / 10001
[t-SNE] Computed conditional probabilities for sample 2000 / 10001
[t-SNE] Computed conditional probabilities for sample 3000 / 10001
[t-SNE] Computed conditional probabilities for sample 4000 / 10001
[t-SNE] Computed conditional probabilities for sample 5000 / 10001
[t-SNE] Computed conditional probabilities for sample 6000 / 10001
```

```
[t-SNE] Computed conditional probabilities for sample 7000 / 10001
[t-SNE] Computed conditional probabilities for sample 8000 / 10001
[t-SNE] Computed conditional probabilities for sample 9000 / 10001
[t-SNE] Computed conditional probabilities for sample 10000 / 10001
[t-SNE] Computed conditional probabilities for sample 10001 / 10001
[t-SNE] Mean sigma: 674.994574
[t-SNE] KL divergence after 250 iterations with early exaggeration: 85.575737
[t-SNE] KL divergence after 300 iterations: 2.941581
```

[13]:
```
# Dimentionality Reduction Techniques using tSNE
import time
N=10000
import pandas as pd
t0 = time.time()
features = ['pixel' + str(i) for i in range(X.shape[1])]
df = pd.DataFrame(X, columns=features)
df['y'] = y
df['label'] = df['y'].apply(lambda i: str(i))


df_tSNE = df.loc[:N, :].copy()



from sklearn.manifold import TSNE
tsne = TSNE(n_components=2) # we're using this value from PCA's␣
 ↪explained_variance_ratio_ value
tsne_res = tsne.fit_transform(df_tSNE[features].values)
tN = time.time()
print('t-SNE time elapsed \t:', tN-t0,  'seconds')
```

```
t-SNE time elapsed      : 37.60014486312866 seconds
```

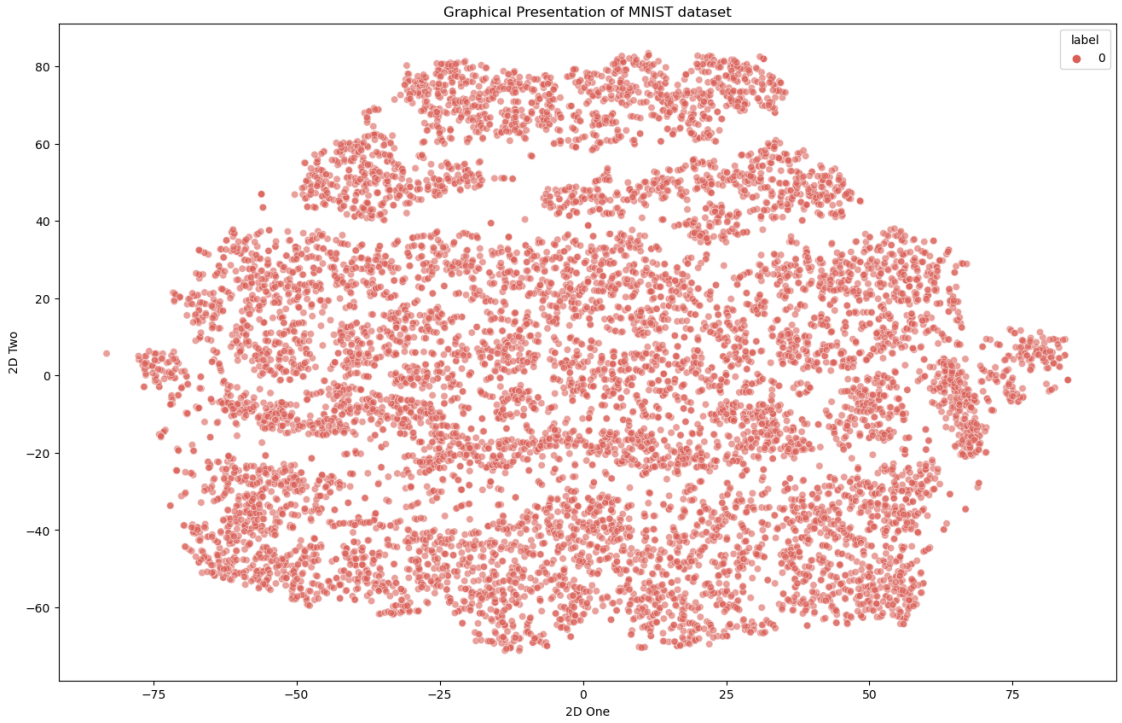
[14]:
```
print(df_tSNE.shape)
#print(tsne_res.shape) ##@## should not be 1
```

```
(10001, 787)
```

[17]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

df_tSNE['2D One'] = tsne_res[:, 0]
df_tSNE['2D Two'] = tsne_res[:, 1]

plt.figure(figsize=(16, 10))
n_labels = df_tSNE['label'].nunique()

sns.scatterplot(
```

```
    x='2D One',
    y='2D Two',
    hue='label',
    palette=sns.color_palette('hls', n_labels),
    data=df_tSNE,
    alpha=0.6
).set(title='Graphical Presentation of MNIST dataset')
```

[17]: [Text(0.5, 1.0, 'Graphical Presentation of MNIST dataset')]



[16]:
```
# Dimentionality Reduction Techniques using LLE
from sklearn.manifold import LocallyLinearEmbedding
t0 = time.time()
lle = LocallyLinearEmbedding(n_components=2,
                             n_neighbors=10,
                             random_state=43)
X_lle = lle.fit_transform(df)
tN = time.time() - t0
print('Time elapsed by LLE: \t', tN, 'seconds.')
```

```
Time elapsed by LLE:     411.7638454437256 seconds.
```

The Locally Linear Embedding (LLE) manifold learning took almost 9 minutes which is high compared to other techniques. This is because LLE is suitable when there are folds in the data, and, it is not so useful in scenarios where we use projections to reduce the dimension of data. We notice

that our dataset has lot of features and it does not have twists. Therefore LLE is not appropriate to apply.