Comp5318_Assignment_2_Random_Forest

November 13, 2020

1 COMP5318 - Machine Learning and Data Mining: Assignment 2 Best Classifier - Random Forest

1.1 Group - 43

Name: Alejandro Diaz SID: 500229318 Nicholas Stollmann SID: 312062796 Name: Shashwati Dutta SID: 500693287

1.2 Introduction

The aim of this assignment is to find the best method to undertake character recognition. The dataset we are analysing is made up of 62992 synthesised characters from computer fonts. The data is split into 62 classes of handwritten images made up of all letters in lower and upper case as well as the ten digits.We propose three classification methods, K- Nearest Neighbor, Random Forest and Convolutional Neural Network and based on series of experiments and evaluation metric we aim to identify the best suited model for the data.

1.3 Libraries

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
import os
from random import sample
import seaborn as sns
import pandas as pd
import time as tm
from PIL import Image, ImageOps
import gc
import string
from collections import Counter, OrderedDict

import h5py


from sklearn.preprocessing import scale, LabelEncoder
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.metrics import confusion_matrix,␣
↪plot_confusion_matrix,accuracy_score,classification_report
```

1.4 Data Loading

The dataset used in our experiments is the The Char47 dataset (English language) which consists of characters in natural images. In this case, we selected the synthesized set of characters. This dataset contains 62992 images of characters from computer fonts. http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/

[4]:
```
######### Raw Data loading #######

#n_dim = 28

#DATAPATH = "./English/Fnt"
#CLASSES = [[str(el) for el in np.arange(0, 10)], list(string.ascii_uppercase),␣
↪list(string.ascii_lowercase)]
#CLASSES = [item for sublist in CLASSES for item in sublist]

#list_imgs = []
#labels = []
#i = 0
 # Get all images
#for folder in sorted(os.listdir(DATAPATH)):
#    folder_content = os.path.join(DATAPATH,folder)
#    for img in sorted(os.listdir(folder_content)):
#        list_imgs.append(os.path.join(folder_content,img))
#        labels.append(CLASSES[i])
#    i += 1

#print(f"Number of images: {len(list_imgs)}")

# # Convert labels list to array
#labels = np.array(labels)

# Iterate over each img
#for i, im in enumerate(list_imgs):

     # Open image
#    image = Image.open(im)
     # Resize it
#    image = image.resize((n_dim, n_dim))
     # If first one - create array
#    if i == 0:
#       images = np.expand_dims(
#             (np.array(image, dtype= float)/255).reshape(-1), axis= 0)
     # Else append images matrix
#    else:
#        image = np.expand_dims(
#             (np.array(image, dtype= float)/255).reshape(-1), axis= 0)
#        images = np.append(images, image, axis= 0)
#    if i % 10000 == 0:
#        print(f"Number of loaded images: {i}")


###### Loading data from .h5 file ######
with h5py.File('./data/images.h5','r') as H:
    images = np.copy(H['images'])
with h5py.File('./data/labels.h5','r') as H:
     labels = np.copy(H['labels'])

print(f"Data shape: {images.shape}")
print(f"Labels shape: {labels.shape}")
```

```
Data shape: (62992, 784)
Labels shape: (62992,)
```

1.5 Preprocessing

In this section, we will implement different preprocess techniques to prepare and improve the quality of our data and thus, improve the performance of our model.

1.5.1 Train, Validation and Test Split

To measure the performance of our algorithms and apply fine-tuning, we will split our data in a training 60%, validation 20% and test set 20%.

[51]:
```
X1, X_test, y1, y_test = train_test_split(images, labels, test_size = 0.20,␣
 ↪random_state = 1) #Test set 20%

#X1, y1 are placeholders for the next split
X_train, X_val, y_train, y_val = train_test_split(X1, y1, test_size = 0.26,␣
 ↪random_state = 1) #Validation set 20% of TOTAL

print(f"X_train shape: {X_train.shape}"),
print(f"X_val shape: {X_val.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_val shape: {y_val.shape}")
print(f"y_test shape: {y_test.shape}")
```

```
X_train shape: (37290, 784)
X_val shape: (13103, 784)
X_test shape: (12599, 784)
y_train shape: (37290,)
```

```
y_val shape: (13103,)
y_test shape: (12599,)
```

1.5.2 Standardize Data

Additionally, an important step in order to be able to apply Principal Component Analysis (PCA) is to standardize our images so that they have mean 0 and variance =1. We can perform this step using the next formula:

$$\frac{x-\mu}{\sigma}$$

However to correctly standardize our data, we should use the training set to calculate the mean and variance, normalize the training set and then normalize the validation and test set using the same mean and variance from training set.

In a nutshell:

scaled_train = (train - train_mean) / train_std_deviation

scaled_validation = (validation - train_mean) / train_std_deviation

scaled_test = (test - train_mean) / train_std_deviation

[52]:
```
train_mean = np.mean(X_train)
train_std_deviation = np.std(X_train)
```

Scaling the training, validation and test sets

[53]:
```
scaled_X_train = (X_train - train_mean) / train_std_deviation
scaled_X_val = (X_val - train_mean) / train_std_deviation
scaled_X_test = (X_test - train_mean) / train_std_deviation
```

1.5.3 Dimensionality Reduction - PCA

Principal components analysis (PCA) is a popular approach for deriving principal components analysis a low-dimensional set of features from a large set of variables. We will use the PCA function from the sklearn package to perform the dimensionality reduction on the scaled dataset.

[54]:
```
n_comp = 256
pca = PCA(n_components = n_comp)

# Fit with our data
X_train = pca.fit_transform(scaled_X_train)
y_train = y_train
```

The shape of our projected data is:

[55]:
```
print(f"Shape of projected data - X_train: {X_train.shape}")
print(f"Shape of y_train: {y_train.shape}")
```

```
Shape of projected data - X_train: (37290, 256)
Shape of y_train: (37290,)
```

Finally, we need to project our validation and test set into these new components.

[56]:
```
X_val = pca.transform(scaled_X_val)
y_val = y_val

X_test = pca.transform(scaled_X_test)
y_test = y_test

print(f"Shape of projected data - X_validation: {X_val.shape}")
print(f"Shape of y_train: {y_val.shape}")
print(f"Shape of projected data - X_test: {X_test.shape}")
print(f"Shape of y_train: {y_test.shape}")
```

```
Shape of projected data - X_validation: (13103, 256)
Shape of y_train: (13103,)
Shape of projected data - X_test: (12599, 256)
Shape of y_train: (12599,)
```

1.6 Random Forest Classifier

This gives the best parameters to use for this random forest: number of estimators = 455 Best Criterion to decide on tree splits = Entropy Maximum number of features = 15 Maximum depth = 180

[22]:
```
start_time = tm.time()

random_forest_best = RandomForestClassifier(n_estimators = 455,criterion =␣
↪"entropy",max_features = 15,max_depth=180, n_jobs = -1)
random_forest_best.fit(X_train,y_train)
print("Best accuracy from Random Forest classifier is {:.2f}%.".
↪format((random_forest_best.score(X_test, y_test)*100)))

print(f"-------Running Time------- {tm.time()-start_time} seconds.")
```

```
Best accuracy from Random Forest classifier is 86.99%.
-------Running Time------- 281.52724385261536 seconds.
```

1.7 Results

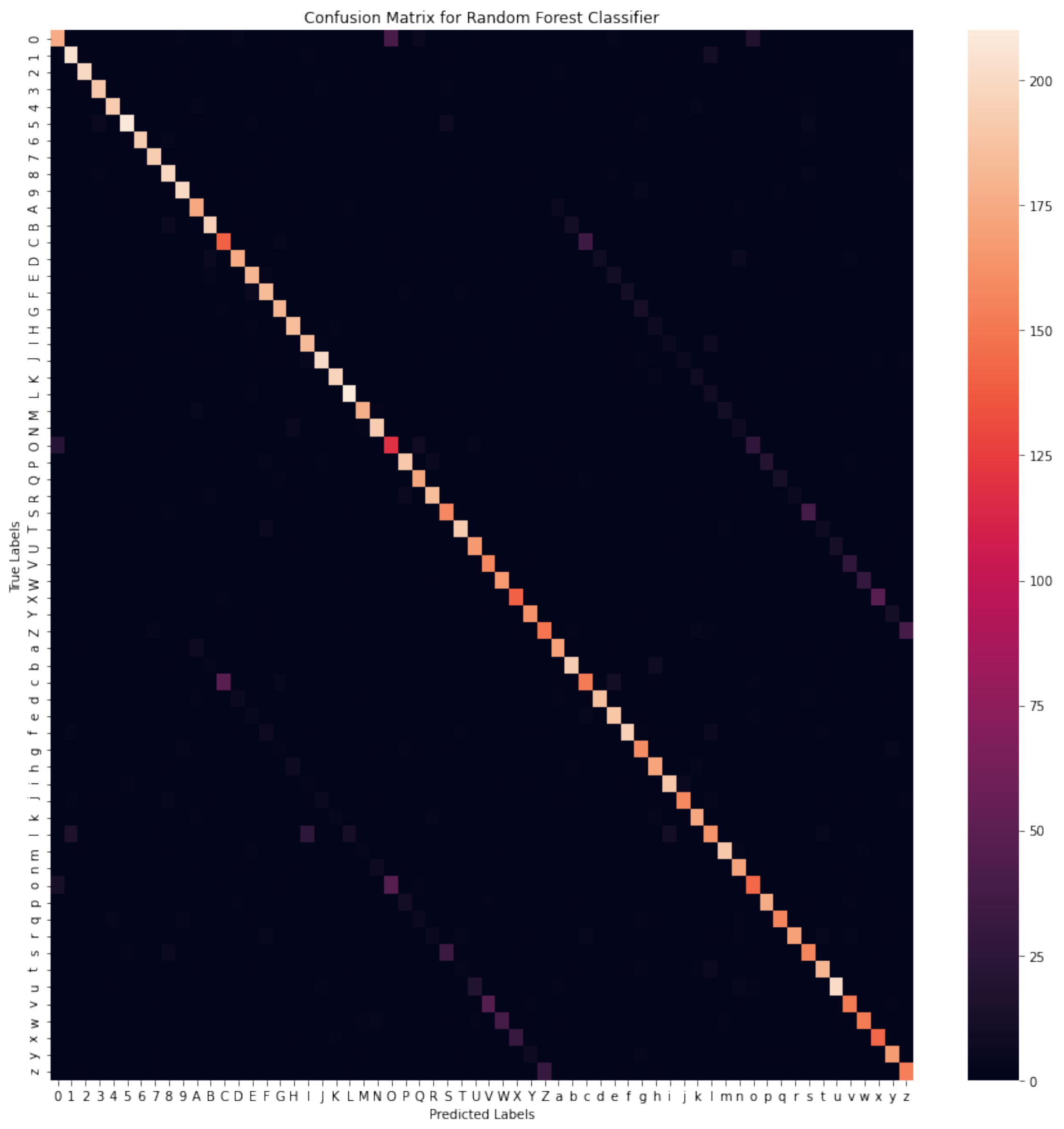
[16]:
```
def result(actual, predicted):

    # confusion matrix
    y_Predicted = np.reshape(predicted, (predicted.shape[0], 1))
    y_Actual = np.reshape(actual, (actual.shape[0], 1))
    df_cm = pd.DataFrame(np.hstack((y_Predicted, y_Actual)),␣
↪columns=['y_Actual','y_Predicted'])
```

```
    confusion_matrix = pd.crosstab(df_cm['y_Actual'], df_cm['y_Predicted'],␣
↪rownames=['Actual'], colnames=['Predicted'])
    print("Confusion Matrix: \n",confusion_matrix)

    # classification report for precision, recall f1-score and accuracy
    matrix = classification_report(actual,predicted, zero_division = 0)
    print('Classification report : \n',matrix)

    plt.figure(figsize = (15,15))
    ax = plt.subplot()
    sns.heatmap(confusion_matrix, ax=ax)
    ax.set_ylabel("True Labels")
    ax.set_xlabel("Predicted Labels")
    ax.set_title("Confusion Matrix for Random Forest Classifier")
    plt.show()
```

[11]:
```
print(result(y_test,random_forest_best.predict(X_test)))
```

```
Confusion Matrix:
 Predicted    0    1    2    3    4  5  6  7  8  9  …  q  r  s  t  u    v  \
Actual                                               …
0          176    0    0    0    0  0  0  0  1  2  …  1  0  0  0  0    0
1            0  205    1    0    0  0  0  0  0  0  …  1  0  0  0  0    0
2            0    1  200    0    0  0  0  1  0  1  …  0  0  0  0  0    0
3            0    0    1  191    0  1  0  0  0  0  …  1  0  1  0  0    0
4            0    0    0    0  192  0  0  0  0  0  …  0  1  2  0  0    0
…          …    …    …    …    … .. .. .. .. ..  … .. .. .. .. ..  …
v            0    0    0    0    0  0  0  1  0  0  …  0  0  0  0  0  152
w            0    0    0    0    0  0  0  0  0  0  …  0  0  0  0  1    0
x            0    0    0    0    0  0  0  0  0  0  …  0  0  0  0  0    0
y            0    0    0    0    0  0  0  1  0  0  …  1  1  0  0  0    0
z            0    0    1    0    0  0  0  1  0  0  …  0  0  0  0  0    0

Predicted    w    x    y    z
Actual
0            0    0    0    0
1            0    0    0    2
2            0    0    0    1
3            0    0    0    0
4            0    0    0    0
…          …    …    …    …
v            1    0    2    0
w          152    0    0    0
x            0  143    0    0
y            0    0  168    0
z            0    0    0  154
```

```
[62 rows x 62 columns]
Classification report :
              precision    recall  f1-score   support

           0       0.70      0.82      0.75       214
           1       0.92      0.88      0.90       232
           2       0.96      0.97      0.97       206
           3       0.93      0.96      0.95       199
           4       0.94      0.95      0.95       202
           5       0.89      0.96      0.92       217
           6       0.94      0.98      0.96       197
           7       0.96      0.96      0.96       201
           8       0.93      0.89      0.91       224
           9       0.96      0.94      0.95       213
           A       0.91      0.87      0.89       199
           B       0.91      0.92      0.91       214
           C       0.78      0.71      0.74       199
           D       0.86      0.91      0.89       193
           E       0.91      0.88      0.89       206
           F       0.89      0.87      0.88       210
           G       0.91      0.91      0.91       199
           H       0.93      0.91      0.92       204
           I       0.88      0.84      0.86       220
           J       0.87      0.92      0.90       219
           K       0.91      0.95      0.93       207
           L       0.94      0.91      0.93       231
           M       0.86      0.95      0.91       187
           N       0.92      0.93      0.93       208
           O       0.65      0.57      0.61       208
           P       0.83      0.90      0.86       213
           Q       0.89      0.87      0.88       199
           R       0.89      0.92      0.90       202
           S       0.79      0.77      0.78       206
           T       0.92      0.94      0.93       204
           U       0.91      0.85      0.88       196
           V       0.83      0.75      0.79       210
           W       0.83      0.80      0.81       210
           X       0.73      0.80      0.76       177
           Y       0.91      0.90      0.91       182
           Z       0.73      0.80      0.77       187
           a       0.91      0.90      0.91       190
           b       0.93      0.92      0.92       210
           c       0.69      0.77      0.73       197
           d       0.92      0.94      0.93       199
           e       0.93      0.86      0.90       220
           f       0.87      0.92      0.89       213
           g       0.91      0.82      0.86       195
           h       0.92      0.86      0.89       201
           i       0.94      0.90      0.92       210
           j       0.85      0.93      0.89       171
           k       0.94      0.89      0.91       196
           l       0.71      0.78      0.74       210
           m       0.94      0.93      0.93       206
           n       0.92      0.83      0.88       206
           o       0.68      0.72      0.70       198
           p       0.90      0.88      0.89       200
           q       0.89      0.88      0.88       179
           r       0.85      0.92      0.89       184
           s       0.79      0.73      0.76       214
           t       0.90      0.90      0.90       202
           u       0.87      0.91      0.89       221
           v       0.74      0.78      0.76       196
           w       0.75      0.80      0.78       189
           x       0.81      0.72      0.76       198
           y       0.90      0.87      0.89       193
           z       0.81      0.75      0.78       206

    accuracy                           0.87     12599
   macro avg       0.87      0.87      0.87     12599
weighted avg       0.87      0.87      0.87     12599
```

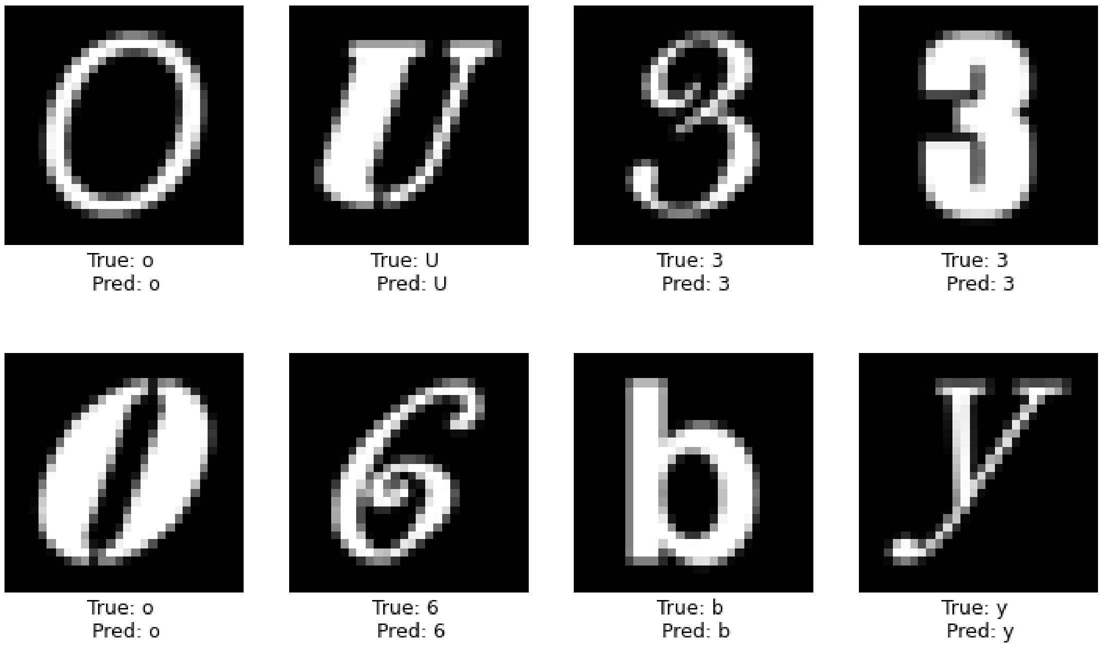


```
None
```

[12]:
```python
def array2img(array, w=28, h=28, c=1):
    """
    Function to convert an 1D array image into a 2D matrix

    :param array: 1D array which we want to convert into a 2D matrix
    :param w: Width image
    :param h: Height image
    :param c: Color channel
    """
    if c==1:
      return np.asarray(array).reshape(w, h)
    else:
      return np.asarray(array).reshape(w, h, c)
```

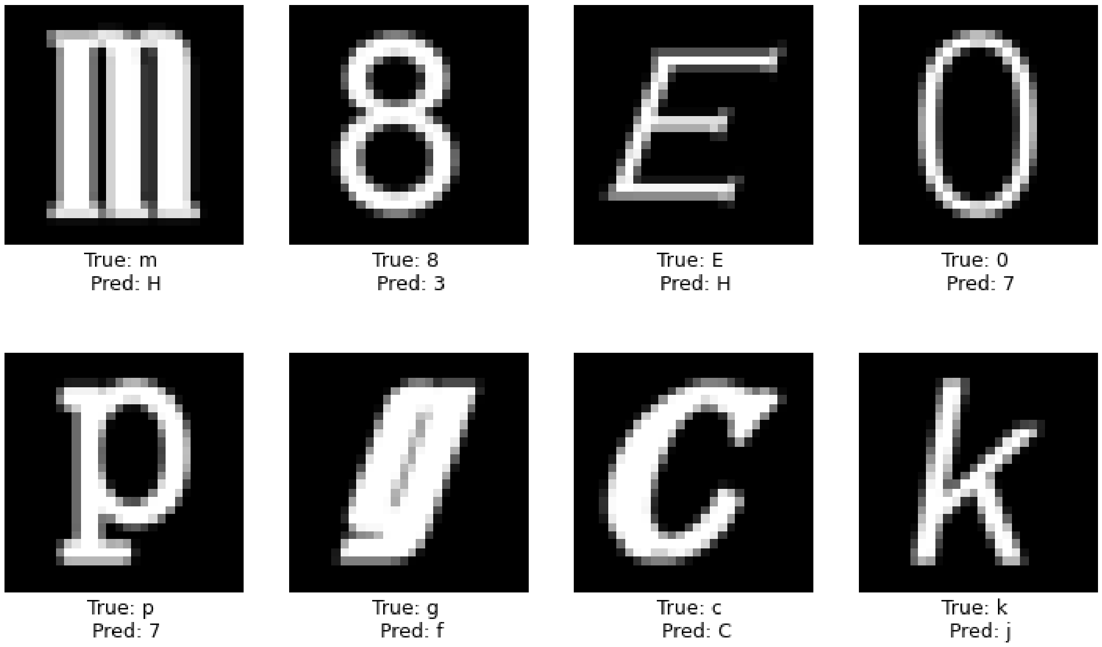
1.7.1 Generating correct predictions

[70]:
```
# Plot correct images
img = [0,1,11,13,14,5,6,7]

plt.figure(figsize=(10, 6), dpi=90)
for i, idx in enumerate(img):
  plt.subplot(2, 4, i+1)
  plt.xticks([])
  plt.yticks([])
  plt.grid(False)
  # Convert each 1d array into 2d matrix
  img2d = array2img(X1[idx])

  # Get label for this image
  true_lbl = y1[idx]
  pred_lbl = random_forest_best.predict(pca.transform([X1[idx]]))[0]
  plt.imshow(img2d, cmap=plt.cm.binary)
  plt.xlabel(f"True: {true_lbl} \n Pred: {pred_lbl}")
plt.show()
```



1.7.2 Generating incorrect predictions

[77]:
```
# Plot incorrect images
img = [2,3,4,21,16,51,30,24]

plt.figure(figsize=(10, 6), dpi=90)
for i, idx in enumerate(img):
  plt.subplot(2, 4, i+1)
  plt.xticks([])
  plt.yticks([])
  plt.grid(False)
  # Convert each 1d array into 2d matrix
  img2d = array2img(X1[idx])

  # Get label for this image
  true_lbl = y1[idx]
  pred_lbl = random_forest_best.predict(pca.transform([X1[idx]]))[0]
  plt.imshow(img2d, cmap=plt.cm.binary)
  plt.xlabel(f"True: {true_lbl} \n Pred: {pred_lbl}")
plt.show()
```



[ ]:
```
```