assignment

September 18, 2024

[22]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

Dataset Link :https://www.kaggle.com/datasets/nikbearbrown/tmnist-alphabet-94-characters

Dataset Deatils: Typography MNIST (TMNIST)

MNIST style images of the following 94 alphabetic characters:

{‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’, ‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, ‘g’, ‘h’, ‘i’, ‘j’, ‘k’, ‘l’, ‘m’, ‘n’, ‘o’, ‘p’, ‘q’, ‘r’, ‘s’, ‘t’, ‘u’, ‘v’, ‘w’, ‘x’, ‘y’, ‘z’, ‘A’, ‘B’, ‘C’, ‘D’, ‘E’, ‘F’, ‘G’, ‘H’, ‘I’, ‘J’, ‘K’, ‘L’, ‘M’, ‘N’, ‘O’, ‘P’, ‘Q’, ‘R’, ‘S’, ‘T’, ‘U’, ‘V’, ‘W’, ‘X’, ‘Y’, ‘Z’, ‘!’, ‘ “ ‘,’#‘,’$‘,’%‘,’&‘, “’ ”,’(‘,’)‘,’*‘,’+‘,’,‘,’-‘,’.’, ‘/’, ‘:’, ‘;’, ‘<’, ‘=’, ‘>’, ‘?’, ‘@’, ‘[’, ‘', ’]’, ‘^’, ’_‘,’“,’{‘,’|‘,’}‘,’~’}

This repository contains a single csv .file. The structure of the csv file is:

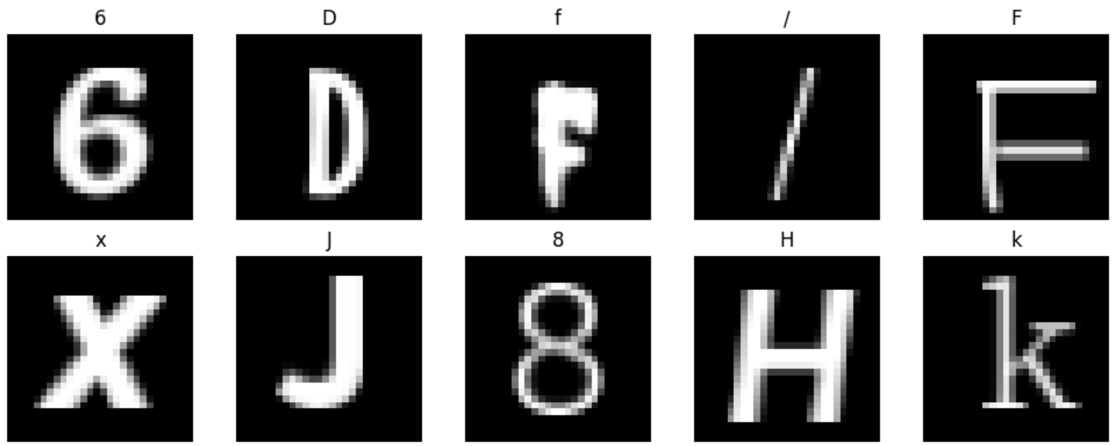
```
the first row contains column headers ['names', 'labels','1','2',…..'784']
The 'names' column contains font file names such as 'Acme-Regular' and 'ZillaSlab-Bold'
The 'labels' column contains characters such as '@','E' or '+'
The remaining 784 columns contain the grayscale pixel values for the image of the correspondin
```

This dataset contains over 281,000 images and is part of the Warhol.ai Computational Creativity and Cognitive Type projects.

[12]:
```python
data = pd.read_csv(r"C:\Users\vishn\Downloads\archive(1)\94_character_TMNIST.
↪csv")
```

[27]:
```python
head = data.head(10)

plt.figure(figsize=(10, 4))
for i in range(10):
    pixels = head.iloc[i, 2:].values.astype(np.uint8).reshape((28, 28))

    label = head.iloc[i, 1]
    font_name = head.iloc[i, 0]

    plt.subplot(2, 5, i+1)
```

```
    plt.imshow(pixels, cmap='gray')
    plt.title(f"{label}")
    plt.axis('off')

plt.tight_layout()
plt.show()
```



Activation Function : Sigmoid

Epochs : 20

Weight Updation Rule : Learning Rate * Xi ,where Xi is the input value

```
[15]: X = data.iloc[:, 2:].values / 255.0
y = data['labels'].values


label_encoder = LabelEncoder()
y_numeric = label_encoder.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y_numeric, test_size=0.
 ↪2, random_state=42)

num_features = X_train.shape[1]
num_classes = len(np.unique(y_numeric))
weights = np.random.randn(num_features, num_classes)
bias = np.zeros(num_classes)
learning_rate = 0.01

def activation(z):
    return 1 / (1 + np.exp(-z))
```

```python
for epoch in range(20):
    for i in range(X_train.shape[0]):
        z = np.dot(X_train[i], weights) + bias
        y_pred = activation(z)

        if y_pred != y_train[i]:
            weights[:, y_train[i]] += learning_rate * X_train[i]
            weights[:, y_pred] -= learning_rate * X_train[i]

z_test = np.dot(X_test, weights) + bias
y_test_pred = activation(z_test)

accuracy = np.mean(y_test_pred == y_test)
print(f"Test accuracy: {accuracy * 100:.2f}%")
```

```
Test accuracy: 80.86%
```

[20]:

```python
cm = confusion_matrix(y_test, y_test_pred)
print("Confusion Matrix:")
print(cm)
print("Classification Report:")
print(classification_report(y_test, y_test_pred, target_names=label_encoder.
 ↪classes_))
```

```
Confusion Matrix:
[[542   0   0 …   3   0   0]
 [  1 520   0 …   0   0   1]
 [  0   0 552 …   0   0   1]
 …
 [  6   0   0 … 504   0   0]
 [  3   0   1 …   0 577   0]
 [  0   1   0 …   0   0 556]]
Classification Report:
              precision    recall  f1-score   support

           !       0.88      0.90      0.89       602
           "       0.88      0.94      0.91       556
           #       0.92      0.95      0.93       579
           $       0.94      0.90      0.92       583
           %       0.94      0.91      0.92       617
           &       0.89      0.88      0.89       566
           '       0.76      0.74      0.75       617
           (       0.96      0.93      0.94       587
           )       0.92      0.93      0.92       591
           *       0.80      0.89      0.84       624
           +       0.98      0.96      0.97       626
           ,       0.75      0.87      0.80       583
```

-	0.68	0.94	0.79	634
.	0.91	0.78	0.84	574
/	0.79	0.90	0.84	597
0	0.49	0.71	0.58	570
1	0.82	0.75	0.78	589
2	0.94	0.93	0.93	613
3	0.93	0.90	0.92	600
4	0.92	0.89	0.91	581
5	0.90	0.83	0.86	588
6	0.95	0.84	0.89	634
7	0.87	0.90	0.89	595
8	0.92	0.84	0.88	586
9	0.89	0.90	0.90	620
:	0.96	0.98	0.97	615
;	0.96	0.93	0.95	584
<	0.96	0.98	0.97	595
=	0.98	0.94	0.96	613
>	0.98	0.96	0.97	622
?	0.97	0.91	0.94	612
@	0.94	0.88	0.91	594
A	0.90	0.82	0.86	547
B	0.83	0.87	0.85	585
C	0.58	0.74	0.65	547
D	0.88	0.89	0.89	571
E	0.87	0.86	0.86	542
F	0.87	0.88	0.87	577
G	0.86	0.78	0.82	600
H	0.90	0.84	0.87	565
I	0.63	0.32	0.42	586
J	0.76	0.79	0.78	576
K	0.86	0.86	0.86	577
L	0.89	0.87	0.88	574
M	0.78	0.88	0.83	565
N	0.72	0.90	0.80	578
O	0.59	0.58	0.58	618
P	0.77	0.89	0.83	604
Q	0.72	0.89	0.79	590
R	0.80	0.92	0.86	595
S	0.65	0.79	0.71	605
T	0.92	0.81	0.86	567
U	0.49	0.90	0.64	615
V	0.53	0.79	0.64	550
W	0.65	0.20	0.31	555
X	0.57	0.78	0.66	602
Y	0.83	0.89	0.86	529
Z	0.69	0.56	0.62	593
[	0.91	0.80	0.85	583
\	0.91	0.93	0.92	576

```
           ]       0.85      0.89      0.87       578
           ^       0.95      0.96      0.96       592
           _       0.90      0.56      0.69       572
           `       0.93      0.92      0.93       550
           a       0.71      0.88      0.79       522
           b       0.89      0.87      0.88       590
           c       0.64      0.51      0.57       587
           d       0.90      0.82      0.86       554
           e       0.80      0.87      0.83       578
           f       0.83      0.79      0.81       588
           g       0.76      0.80      0.78       630
           h       0.85      0.85      0.85       598
           i       0.91      0.77      0.83       558
           j       0.78      0.84      0.81       601
           k       0.94      0.81      0.87       594
           l       0.58      0.38      0.46       540
           m       0.91      0.87      0.89       565
           n       0.91      0.79      0.85       575
           o       0.68      0.35      0.46       551
           p       0.89      0.72      0.80       578
           q       0.86      0.83      0.84       567
           r       0.90      0.78      0.84       554
           s       0.66      0.61      0.63       561
           t       0.64      0.91      0.75       588
           u       0.85      0.23      0.37       614
           v       0.68      0.25      0.37       571
           w       0.50      0.85      0.63       503
           x       0.71      0.43      0.54       602
           y       0.89      0.81      0.85       570
           z       0.60      0.64      0.62       501
           {       0.96      0.95      0.95       588
           |       0.50      0.87      0.64       578
           }       0.95      0.93      0.94       619
           ~       0.97      0.95      0.96       583

    accuracy                           0.81     54819
   macro avg       0.82      0.81      0.80     54819
weighted avg       0.82      0.81      0.80     54819
```

[25]:
```
y_test_actual_labels = label_encoder.inverse_transform(y_test)
y_test_pred_labels = label_encoder.inverse_transform(y_test_pred)

results_df = pd.DataFrame({
    'Actual': y_test_actual_labels,
    'Predicted': y_test_pred_labels,
    'Correct': y_test_actual_labels == y_test_pred_labels
```

```
})
print(results_df.head(10))
```

```
  Actual Predicted  Correct
0      &         &     True
1      '         '     True
2      i         i     True
3      v         V    False
4      /         /     True
5      _         _     True
6      S         d    False
7      g         ,    False
8      ,         ,     True
9      h         H    False
```

Save the trained weights so you dont have to train the model again and again.

[26]:
```
np.savez('perceptron_model_weights.npz', weights=weights, bias=bias)
```