Mnist digit recognition with csv data

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

train_df = pd.read_csv('mnist_train.csv')
test_df = pd.read_csv('mnist_test.csv')
train_df.head()
```

```
   label  1x1  1x2  1x3  1x4  1x5  1x6  1x7  1x8  1x9  ...  28x19
28x20  \
0      5    0    0    0    0    0    0    0    0    0  ...       0
0
1      0    0    0    0    0    0    0    0    0    0  ...       0
0
2      4    0    0    0    0    0    0    0    0    0  ...       0
0
3      1    0    0    0    0    0    0    0    0    0  ...       0
0
4      9    0    0    0    0    0    0    0    0    0  ...       0
0

   28x21  28x22  28x23  28x24  28x25  28x26  28x27  28x28
0      0      0      0      0      0      0      0      0
1      0      0      0      0      0      0      0      0
2      0      0      0      0      0      0      0      0
3      0      0      0      0      0      0      0      0
4      0      0      0      0      0      0      0      0

[5 rows x 785 columns]
```

```
test_df.head()
```

```
   label  1x1  1x2  1x3  1x4  1x5  1x6  1x7  1x8  1x9  ...  28x19
28x20  \
0      7    0    0    0    0    0    0    0    0    0  ...       0
0
1      2    0    0    0    0    0    0    0    0    0  ...       0
0
2      1    0    0    0    0    0    0    0    0    0  ...       0
0
3      0    0    0    0    0    0    0    0    0    0  ...       0
0
4      4    0    0    0    0    0    0    0    0    0  ...       0
0
```

```
   28x21  28x22  28x23  28x24  28x25  28x26  28x27  28x28
0      0      0      0      0      0      0      0      0
1      0      0      0      0      0      0      0      0
2      0      0      0      0      0      0      0      0
3      0      0      0      0      0      0      0      0
4      0      0      0      0      0      0      0      0

[5 rows x 785 columns]
```

```
train_df.shape
```

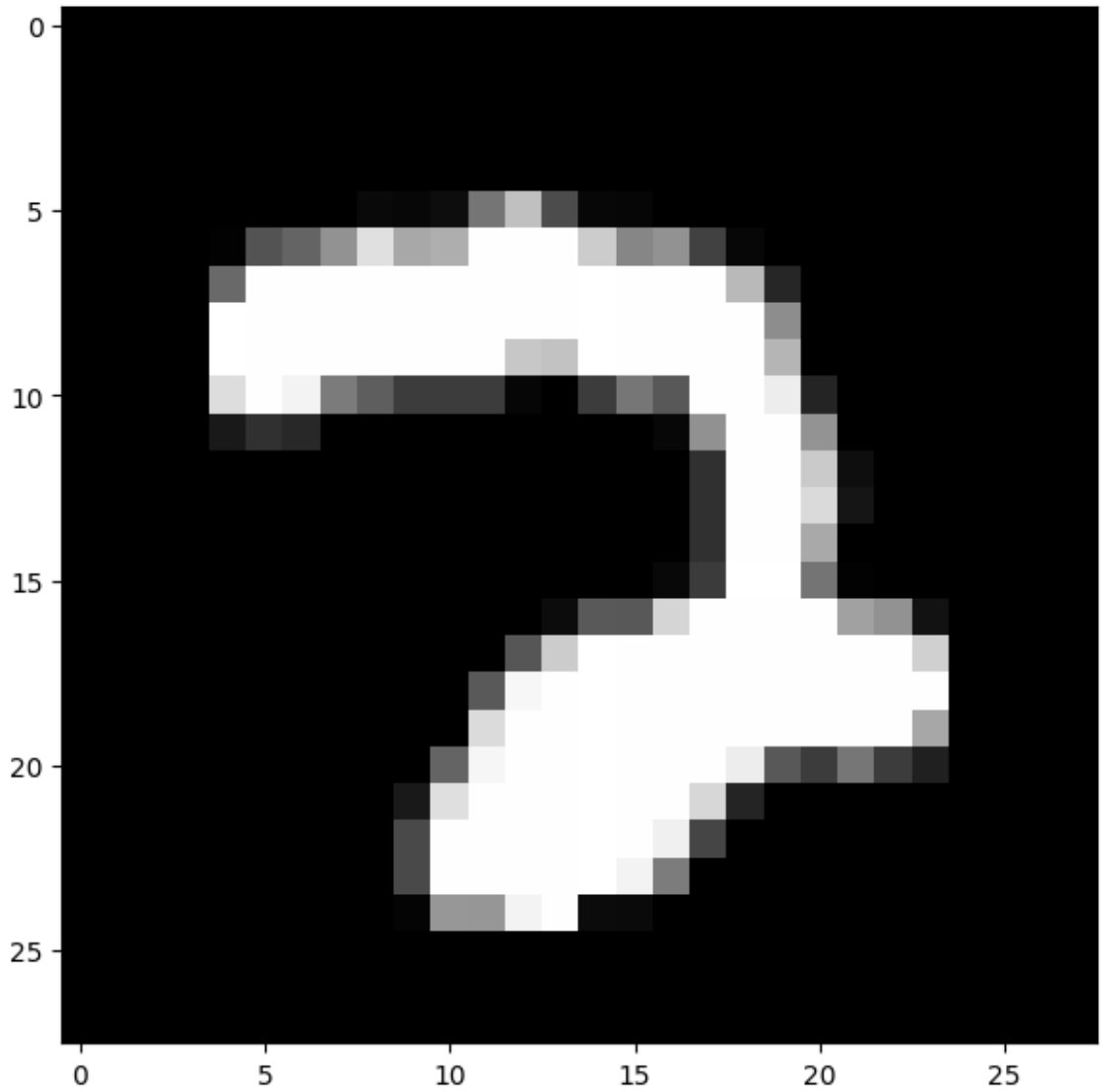
```
(60000, 785)
```

```
y = train_df['label']
x = train_df.drop('label',axis=1)

x_for_test_data = test_df[:]

plt.figure(figsize = (7,7))
some_digit =120
some_digit_image = x.iloc[some_digit].to_numpy().reshape(28,28)
plt.imshow(np.reshape(some_digit_image,(28,28)),cmap=plt.cm.gray)
print(y[some_digit])
```

```
2
```




```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size =
0.30,random_state =42)
```

KNN

```
x_train.shape,y_train.shape,x_test.shape,y_test.shape
```

```
((42000, 784), (42000,), (18000, 784), (18000,))
```

```
from sklearn.preprocessing import StandardScaler
scaler =  StandardScaler()
scaler.fit(x_train,y_train)
```

```
x_train = scaler.transform(x_train)
x_train.shape
```

```
(42000, 784)
```

k=5

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5)
classifier.fit(x_train,y_train)
```

```
KNeighborsClassifier()
```

```
y_pred = classifier.predict(x_test)
y_pred
```

```
C:\Users\Dell\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\base.py:402: UserWarning: X has feature names, but
KNeighborsClassifier was fitted without feature names
  warnings.warn(
```

```
array([7, 3, 8, ..., 5, 0, 0], dtype=int64)
```

```
from sklearn.metrics import accuracy_score,
classification_report,confusion_matrix
print(accuracy_score(y_test,y_pred))
print(classification_report(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
```

```
0.8108888888888889
              precision    recall  f1-score   support

           0       0.61      0.98      0.75      1805
           1       1.00      0.87      0.93      1994
           2       0.91      0.86      0.89      1759
           3       0.93      0.73      0.82      1846
           4       0.98      0.61      0.75      1726
           5       0.98      0.57      0.72      1653
           6       0.96      0.92      0.94      1787
           7       0.96      0.82      0.89      1937
           8       0.51      0.94      0.66      1730
           9       0.84      0.77      0.80      1763

    accuracy                           0.81     18000
   macro avg       0.87      0.81      0.81     18000
weighted avg       0.87      0.81      0.82     18000

[[1774    0    0    0    0    0    4    0   27    0]
 [   4 1744   26    1    2    1    9    4  201    2]
```

```
 [ 157    0 1512   14    1    0    6    3   63    3]
 [ 159    1   60 1343    0    9    1   10  256    7]
 [ 278    0    3    2 1047    1   27    4  212  152]
 [ 239    0    1   52    0  942   26    1  387    5]
 [  88    0    0    0    1    3 1643    0   52    0]
 [  52    0   38   15    9    0    0 1597  134   92]
 [  60    3   12    8    2    9    3    0 1633    0]
 [ 115    1    1    2    2    1    0   41  239 1361]]
```

```
# SVM

# create a svm classifier
#from sklearn import svm
#clf = svm.SVC(kernel ='linear', C=10,random_state =0) # linear kernel

# train the model using training sets
#clf.fit(x_train,y_train)

#predict the response for test dataset
#y_pred_svm = clf.predict(x_test)
#y_pred_svm

#from sklearn.metrics import accuracy_score,
classification_report,confusion_matrix
#print(accuracy_score(y_test,y_pred_svm))
#print(classification_repot(y_test,y_pred_svm))
#print(confusion_matrix(y_test,y_pred_svm))
```

k=7

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 7)
classifier.fit(x_train,y_train)
```

```
KNeighborsClassifier(n_neighbors=7)
```

```
y_pred = classifier .predict(x_test)
y_pred
```

```
C:\Users\Dell\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\base.py:402: UserWarning: X has feature names, but
KNeighborsClassifier was fitted without feature names
  warnings.warn(
```

```
array([7, 3, 8, ..., 5, 0, 0], dtype=int64)
```

```
from sklearn.metrics import
accuracy_score,classification_report,confusion_matrix
print(accuracy_score(y_test,y_pred))
```

```
print(classification_report(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
```

```
0.8176111111111111
              precision    recall  f1-score   support

           0       0.63      0.99      0.77      1805
           1       1.00      0.89      0.94      1994
           2       0.92      0.87      0.89      1759
           3       0.94      0.75      0.84      1846
           4       0.98      0.61      0.75      1726
           5       0.99      0.56      0.71      1653
           6       0.96      0.92      0.94      1787
           7       0.97      0.82      0.89      1937
           8       0.51      0.95      0.66      1730
           9       0.84      0.79      0.81      1763

    accuracy                           0.82     18000
   macro avg       0.87      0.81      0.82     18000
weighted avg       0.88      0.82      0.82     18000

[[1779    0    0    0    0    0    4    0   22    0]
 [   4 1779   27    2    2    0    6    4  169    1]
 [ 139    1 1523   11    2    0    5    2   73    3]
 [ 128    1   44 1385    0    5    0    9  268    6]
 [ 257    0    4    1 1055    0   25    5  228  151]
 [ 231    0    2   45    0  925   31    1  411    7]
 [  82    0    0    0    1    2 1644    0   58    0]
 [  63    0   39   16    8    0    0 1593  130   88]
 [  52    3    9    6    3    5    2    0 1650    0]
 [ 101    1    0    3    3    1    0   33  237 1384]]
```

k=23

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 23)
classifier.fit(x_train,y_train)
```

```
KNeighborsClassifier(n_neighbors=23)
```

```
y_pred = classifier .predict(x_test)
y_pred
```

```
C:\Users\Dell\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\base.py:402: UserWarning: X has feature names, but
KNeighborsClassifier was fitted without feature names
  warnings.warn(

array([7, 3, 8, ..., 5, 0, 0], dtype=int64)
```

```
from sklearn.metrics import
accuracy_score,classification_report,confusion_matrix
print(accuracy_score(y_test,y_pred))
print(classification_report(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
```

```
0.8309444444444445
              precision    recall  f1-score   support

           0       0.68      0.99      0.80      1805
           1       1.00      0.91      0.95      1994
           2       0.93      0.89      0.91      1759
           3       0.94      0.80      0.86      1846
           4       0.99      0.61      0.75      1726
           5       1.00      0.53      0.70      1653
           6       0.95      0.94      0.94      1787
           7       0.97      0.83      0.90      1937
           8       0.52      0.96      0.67      1730
           9       0.84      0.81      0.82      1763

    accuracy                           0.83     18000
   macro avg       0.88      0.83      0.83     18000
weighted avg       0.88      0.83      0.84     18000

[[1783    0    0    0    0    0    4    0   18    0]
 [   3 1820   22    3    2    0    9    1  133    1]
 [ 104    0 1574   13    1    0    5    1   59    2]
 [  89    1   37 1480    0    1    1    7  225    5]
 [ 174    0    3    1 1048    0   45    4  273  178]
 [ 234    0    2   65    0  884   28    1  435    4]
 [  78    0    0    0    1    2 1676    0   30    0]
 [  53    0   41   10    3    0    0 1611  132   87]
 [  46    2    8    6    1    1    4    0 1661    1]
 [  73    0    0    3    1    0    1   35  230 1420]]
```

SVM

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

#importing dataset
dataset_train=pd.read_csv('mnist_train.csv') # has 10 classes-- 0 to 9
digits and labels
dataset_test=pd.read_csv('mnist_test.csv')
```

```
print("Shape of training dataset:",dataset_train.shape)
print("Shape of testing dataset:",dataset_test.shape)
```

```
Shape of training dataset: (60000, 785)
Shape of testing dataset: (10000, 785)
```

```
print("Description of training dataset:",dataset_train.describe())
print("Description of test dataset:",dataset_test.describe())
```

```
Description of training dataset:                label      1x1      1x2
1x3      1x4      1x5      1x6  \
count  60000.000000  60000.0  60000.0  60000.0  60000.0  60000.0
60000.0
mean       4.453933      0.0      0.0      0.0      0.0      0.0
0.0
std        2.889270      0.0      0.0      0.0      0.0      0.0
0.0
min        0.000000      0.0      0.0      0.0      0.0      0.0
0.0
25%        2.000000      0.0      0.0      0.0      0.0      0.0
0.0
50%        4.000000      0.0      0.0      0.0      0.0      0.0
0.0
75%        7.000000      0.0      0.0      0.0      0.0      0.0
0.0
max        9.000000      0.0      0.0      0.0      0.0      0.0
0.0

           1x7      1x8      1x9  ...         28x19         28x20  \
count  60000.0  60000.0  60000.0  ...  60000.000000  60000.000000
mean       0.0      0.0      0.0  ...      0.200433      0.088867
std        0.0      0.0      0.0  ...      6.042472      3.956189
min        0.0      0.0      0.0  ...      0.000000      0.000000
25%        0.0      0.0      0.0  ...      0.000000      0.000000
50%        0.0      0.0      0.0  ...      0.000000      0.000000
75%        0.0      0.0      0.0  ...      0.000000      0.000000
max        0.0      0.0      0.0  ...    254.000000    254.000000

              28x21         28x22         28x23       28x24    28x25
28x26  \
count  60000.000000  60000.000000  60000.000000  60000.0000  60000.0
60000.0
mean       0.045633      0.019283      0.015117      0.0020      0.0
0.0
std        2.839845      1.686770      1.678283      0.3466      0.0
0.0
min        0.000000      0.000000      0.000000      0.0000      0.0
0.0
25%        0.000000      0.000000      0.000000      0.0000      0.0
0.0
```

```
50%        0.000000      0.000000      0.000000      0.0000      0.0
0.0
75%        0.000000      0.000000      0.000000      0.0000      0.0
0.0
max      253.000000    253.000000    254.000000     62.0000      0.0
0.0

         28x27    28x28
count  60000.0  60000.0
mean       0.0      0.0
std        0.0      0.0
min        0.0      0.0
25%        0.0      0.0
50%        0.0      0.0
75%        0.0      0.0
max        0.0      0.0

[8 rows x 785 columns]
Description of test dataset:                label      1x1      1x2
1x3      1x4      1x5      1x6  \
count  10000.000000  10000.0  10000.0  10000.0  10000.0  10000.0
10000.0
mean       4.443400      0.0      0.0      0.0      0.0      0.0
0.0
std        2.895865      0.0      0.0      0.0      0.0      0.0
0.0
min        0.000000      0.0      0.0      0.0      0.0      0.0
0.0
25%        2.000000      0.0      0.0      0.0      0.0      0.0
0.0
50%        4.000000      0.0      0.0      0.0      0.0      0.0
0.0
75%        7.000000      0.0      0.0      0.0      0.0      0.0
0.0
max        9.000000      0.0      0.0      0.0      0.0      0.0
0.0

           1x7      1x8      1x9  ...         28x19         28x20  \
count  10000.0  10000.0  10000.0  ...  10000.000000  10000.000000
mean       0.0      0.0      0.0  ...      0.179300      0.163600
std        0.0      0.0      0.0  ...      5.674149      5.736072
min        0.0      0.0      0.0  ...      0.000000      0.000000
25%        0.0      0.0      0.0  ...      0.000000      0.000000
50%        0.0      0.0      0.0  ...      0.000000      0.000000
75%        0.0      0.0      0.0  ...      0.000000      0.000000
max        0.0      0.0      0.0  ...    253.000000    253.000000

              28x21       28x22    28x23    28x24    28x25    28x26
28x27  \
count  10000.000000  10000.0000  10000.0  10000.0  10000.0  10000.0
```

```
10000.0
mean       0.052600      0.0006      0.0      0.0      0.0      0.0
0.0
std        2.420004      0.0600      0.0      0.0      0.0      0.0
0.0
min        0.000000      0.0000      0.0      0.0      0.0      0.0
0.0
25%        0.000000      0.0000      0.0      0.0      0.0      0.0
0.0
50%        0.000000      0.0000      0.0      0.0      0.0      0.0
0.0
75%        0.000000      0.0000      0.0      0.0      0.0      0.0
0.0
max      156.000000      6.0000      0.0      0.0      0.0      0.0
0.0

         28x28
count  10000.0
mean       0.0
std        0.0
min        0.0
25%        0.0
50%        0.0
75%        0.0
max        0.0

[8 rows x 785 columns]
```

```
#checking for null and np.nan values
print("Null values in training
dataset:",dataset_train.isnull().sum().head(10))
print("Null values in test
dataset:",dataset_test.isnull().sum().head(10))
```

```
Null values in training dataset: label    0
1x1      0
1x2      0
1x3      0
1x4      0
1x5      0
1x6      0
1x7      0
1x8      0
1x9      0
dtype: int64
Null values in test dataset: label    0
1x1      0
1x2      0
1x3      0
1x4      0
```

```
1x5      0
1x6      0
1x7      0
1x8      0
1x9      0
dtype: int64
```

```
order = list(np.sort(dataset_train['label'].unique()))
print(order)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
# average feature values
round(dataset_train.drop('label', axis=1).mean(), 2)
#drop label column of dataset while calculating mean of all other
columns rounded off upto 2nd
```

```
1x1      0.0
1x2      0.0
1x3      0.0
1x4      0.0
1x5      0.0
        ...
28x24    0.0
28x25    0.0
28x26    0.0
28x27    0.0
28x28    0.0
Length: 784, dtype: float64
```

```
# Separating the X and Y variable

y_dataset_train = dataset_train['label']
# Dropping the variable 'label' from X variable
X_dataset_train = dataset_train.drop(columns = 'label') #drop  labels
column at index 0
# Printing the size of data
print(X_dataset_train.shape)
print(y_dataset_train.shape)
```

```
(60000, 784)
(60000,)
```

```
# Normalization-- all values lie between 0 and 255 so by dividing all
values by 255, they will lie between 0 and 1
X_dataset_train = X_dataset_train/255.0
dataset_test = dataset_test/255.0
print("X:",X_dataset_train.shape)
print("Test dataset:",dataset_test.shape)
```

```
X: (60000, 784)
Test dataset: (10000, 785)
```

```
from sklearn.preprocessing import MinMaxScaler
#feature scaling independent feature variable X
mm_X = MinMaxScaler()
X_dataset_train = mm_X.fit_transform(X_dataset_train)

#splitting dataset into training set and test set
from sklearn.model_selection import train_test_split
# train test split
X_train, X_test, y_train, y_test = train_test_split(X_dataset_train,
y_dataset_train, test_size = 0.2,random_state = 0)

# Applying SVM model to training dataset
from sklearn.svm import SVC
classifier=SVC(kernel='rbf') #hyperparameter tuning c and gamma values
#linear kernel only gives 91.8% accuracy
#polynomial kernel with degree=3 gives 86% accuracy
#sigmoid kernel gives 91.1% accuracy
#rbf kernel gives 97.816% accuracy so i used rbf kernel
model = classifier.fit(X_train,y_train)

#Predicting results of training dataset
y_pred=classifier.predict(X_test)
print("Predicted output values:",y_pred)
```

```
Predicted output values: [3 6 6 ... 5 1 6]
```

```
y_pred.flatten()
```

```
array([3, 6, 6, ..., 5, 1, 6], dtype=int64)
```

```
y_pred.shape
```

```
(12000,)
```

```
# confusion matrix and accuracy

from sklearn import metrics
from sklearn.metrics import confusion_matrix

accuracy=metrics.accuracy_score(y_true=y_test, y_pred=y_pred)*100
print("Accuracy:", accuracy, "\n")

# cm
print(metrics.confusion_matrix(y_true=y_test, y_pred=y_pred))

# CALCULTING ERROR AND ACCURACY OF PREDICTION MODEL
from sklearn.metrics import
mean_absolute_error,mean_squared_error,r2_score
mse = mean_squared_error(y_test, y_pred)
```

```
print('Mean squared error:',mse)
mae = mean_absolute_error(y_test, y_pred)
print('Mean absolute error:',mae)
r_score = r2_score(y_test, y_pred)
print('Accuracy:',r_score*100)
```

```
Accuracy: 97.81666666666666
```

```
[[1195    0    0    0    0    2    6    0    2    0]
 [   0 1372    2    1    2    0    1    0    0    1]
 [   0    1 1142    5    4    1    1    6    5    1]
 [   0    0    9 1169    0   14    0    4   12    0]
 [   0    2    2    0 1132    0    3    4    1    9]
 [   5    0    3   12    4 1035    8    0    5    3]
 [   2    2    3    0    3    5 1173    0    2    0]
 [   0    2    8    1    5    0    1 1199    1   11]
 [   1    5    5    2    2    8    4    0 1161    3]
 [   1    0    0    6   17    5    0   10    6 1160]]
Mean squared error: 0.3388333333333333
Mean absolute error: 0.0755
Accuracy: 95.98114200470734
```

```
from sklearn.metrics import
accuracy_score,classification_report,confusion_matrix
print(accuracy_score(y_test,y_pred))
print(classification_report(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
```

```
0.9781666666666666
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1205
           1       0.99      0.99      0.99      1379
           2       0.97      0.98      0.98      1166
           3       0.98      0.97      0.97      1208
           4       0.97      0.98      0.98      1153
           5       0.97      0.96      0.97      1075
           6       0.98      0.99      0.98      1190
           7       0.98      0.98      0.98      1228
           8       0.97      0.97      0.97      1191
           9       0.98      0.96      0.97      1205

    accuracy                           0.98     12000
   macro avg       0.98      0.98      0.98     12000
weighted avg       0.98      0.98      0.98     12000

[[1195    0    0    0    0    2    6    0    2    0]
 [   0 1372    2    1    2    0    1    0    0    1]
 [   0    1 1142    5    4    1    1    6    5    1]
 [   0    0    9 1169    0   14    0    4   12    0]
```

```
[   0    2    2    0 1132    0    3    4    1    9]
[   5    0    3   12    4 1035    8    0    5    3]
[   2    2    3    0    3    5 1173    0    2    0]
[   0    2    8    1    5    0    1 1199    1   11]
[   1    5    5    2    2    8    4    0 1161    3]
[   1    0    0    6   17    5    0   10    6 1160]]
```

Random Forest

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.ensemble import RandomForestClassifier
#from sklearn.cross_validation import train_test_split
%matplotlib inline

data = pd.read_csv('mnist_train.csv')
#test_df = pd.read_csv('mnist_test.csv')
data.head()
```

```
   label  1x1  1x2  1x3  1x4  1x5  1x6  1x7  1x8  1x9  ...  28x19
28x20  \
0      5    0    0    0    0    0    0    0    0    0  ...       0
0
1      0    0    0    0    0    0    0    0    0    0  ...       0
0
2      4    0    0    0    0    0    0    0    0    0  ...       0
0
3      1    0    0    0    0    0    0    0    0    0  ...       0
0
4      9    0    0    0    0    0    0    0    0    0  ...       0
0

   28x21  28x22  28x23  28x24  28x25  28x26  28x27  28x28
0      0      0      0      0      0      0      0      0
1      0      0      0      0      0      0      0      0
2      0      0      0      0      0      0      0      0
3      0      0      0      0      0      0      0      0
4      0      0      0      0      0      0      0      0

[5 rows x 785 columns]
```

```
a= data.iloc[3,1:].values

df_x = data.iloc[:,1:]
df_y = data.iloc[:,0]
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(df_x,df_y,test_size =
0.30,random_state =42)

x_train.head()
```

```
       1x1  1x2  1x3  1x4  1x5  1x6  1x7  1x8  1x9  1x10  ...  28x19
28x20  \
918      0    0    0    0    0    0    0    0    0     0  ...      0
0
17141    0    0    0    0    0    0    0    0    0     0  ...      0
0
15558    0    0    0    0    0    0    0    0    0     0  ...      0
0
27327    0    0    0    0    0    0    0    0    0     0  ...      0
0
11606    0    0    0    0    0    0    0    0    0     0  ...      0
0

       28x21  28x22  28x23  28x24  28x25  28x26  28x27  28x28
918        0      0      0      0      0      0      0      0
17141      0      0      0      0      0      0      0      0
15558      0      0      0      0      0      0      0      0
27327      0      0      0      0      0      0      0      0
11606      0      0      0      0      0      0      0      0

[5 rows x 784 columns]
```

```
y_train.head()
```

```
918      7
17141    4
15558    5
27327    0
11606    1
Name: label, dtype: int64
```

```
rf = RandomForestClassifier(n_estimators=100)
rf.fit(x_train,y_train)
```

```
RandomForestClassifier()
```

```
pred = rf.predict(x_test)
pred
```

```
array([7, 3, 8, ..., 5, 0, 0], dtype=int64)
```

```
s = y_test.values
count=0
```

```
for i in range(len(pred)):
    if pred[i] == s[i]:
        count = count+1

count
```

```
17379
```

```
len(pred)
```

```
18000
```

```
17383/18000.0
```

```
0.9657222222222223
```

```
from sklearn.metrics import
accuracy_score,classification_report,confusion_matrix
print(accuracy_score(y_test,pred))
print(classification_report(y_test,pred))
print(confusion_matrix(y_test,pred))
```

```
0.9655
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      1805
           1       0.98      0.98      0.98      1994
           2       0.95      0.97      0.96      1759
           3       0.96      0.94      0.95      1846
           4       0.96      0.97      0.97      1726
           5       0.96      0.96      0.96      1653
           6       0.98      0.98      0.98      1787
           7       0.97      0.96      0.97      1937
           8       0.96      0.94      0.95      1730
           9       0.95      0.95      0.95      1763

    accuracy                           0.97     18000
   macro avg       0.97      0.97      0.97     18000
weighted avg       0.97      0.97      0.97     18000

[[1775    0    2    2    3    4    7    0    9    3]
 [   0 1964   11    7    4    0    2    3    1    2]
 [   4    7 1709    6    5    1    2   16    6    3]
 [   2    1   29 1742    1   28    1   18   14   10]
 [   2    1    2    0 1672    1    6    4    3   35]
 [  10    4    3   17    2 1585   12    1   13    6]
 [   5    1    3    0    5   11 1756    0    6    0]
 [   1    9   26    0    8    2    0 1868    5   18]
 [   1    8   15   17   15   17    8    3 1633   13]
 [   8    4    6   17   21    7    2   14    9 1675]]
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
log_classifier = LogisticRegression(random_state=1000)
log_classifier.fit(x_train,y_train)
import warnings
filter.warnings('ignore')
```

```
C:\Users\Dell\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
LogisticRegression(random_state=1000)
```

```
y_pred_log = log_classifier.predict(x_test)
y_pred_log
```

```
array([7, 3, 8, ..., 5, 0, 0], dtype=int64)
```

```
print(accuracy_score(y_test,y_pred_log))
print(classification_report(y_test,y_pred_log))
print(confusion_matrix(y_test,y_pred_log))
```

```
0.9207777777777778
              precision    recall  f1-score   support

           0       0.95      0.96      0.96      1805
           1       0.96      0.97      0.97      1994
           2       0.91      0.89      0.90      1759
           3       0.90      0.89      0.90      1846
           4       0.93      0.94      0.93      1726
           5       0.90      0.87      0.88      1653
           6       0.95      0.95      0.95      1787
           7       0.94      0.92      0.93      1937
           8       0.88      0.89      0.88      1730
           9       0.89      0.91      0.90      1763

    accuracy                           0.92     18000
   macro avg       0.92      0.92      0.92     18000
weighted avg       0.92      0.92      0.92     18000
```

```
[[1734    1    8    6    9   14   12    4   13    4]
 [   0 1941   12    9    2    5    1    3   16    5]
 [   5   20 1574   28   17    8   28   24   45   10]
 [   9   11   47 1640    1   57    3   17   40   21]
 [   5    4    8    3 1614    4   15    4   10   59]
 [  21    9   17   61   10 1440   22    5   60    8]
 [  11    1   22    1   18   20 1700    1   12    1]
 [   6    8   25   14   14    3    0 1791    5   71]
 [  15   28   19   39   14   41   16    5 1538   15]
 [  10    5    7   17   41   11    0   53   17 1602]]
```

Summary

KNN

K=5 with accuracy 81.1%

K=7 with accuracy 81.7%

K=23 with accuracy 83.1%

Logistic Regression with accuracy 92.1%

Random Forest with accuracy 96.6%

SVM with accuracy 97.8%