

LAPORAN

Penerapan KNN, Naive Bayes, dan Multilayer Perceptron terhadap dataset "MNIST Handwritten Digit Database"

Mata Kuliah Kecerdasan Buatan



Nama : Iyan Zuli Armanda

NIM : 23051204165

Kelas : TI 23 E

S1 - Teknik Informatika
Teknik
Universitas Negeri Surabaya
2024

PENDAHULUAN

A. Tugas

- Penerapan KNN, Naive Bayes, dan Multilayer Perceptron terhadap dataset "MNIST Handwritten Digit Database"
- Gunakan salahsatu dataset dari:
 - <https://github.com/cvdfoundation/mnist>
 - <https://huggingface.co/datasets/ylecun/mnist>
- Buat laporan perbandingan hasil ketiga metode Naive Bayes sudah ada di PPT, tambahkan KNN dan Multilayer Perceptron

B. Pengertian

1. Naive Bayes

Naive Bayes adalah algoritma klasifikasi yang didasarkan pada teorema Bayes dengan asumsi bahwa setiap fitur dalam data bersifat independen satu sama lain (naive). Algoritma ini menghitung probabilitas terjadinya suatu kelas berdasarkan nilai fitur-fitur input, kemudian memilih kelas dengan probabilitas tertinggi sebagai hasil prediksi. Naive Bayes sering digunakan dalam aplikasi seperti klasifikasi teks (misalnya, spam email) dan analisis sentimen karena kesederhanaannya dan kemampuannya untuk bekerja dengan data besar secara efisien

2. KNN

KNN (K-Nearest Neighbors) adalah algoritma klasifikasi dan regresi yang sederhana, di mana prediksi dilakukan berdasarkan kedekatan data dengan titik-titik data lainnya. Untuk klasifikasi, KNN menentukan kelas suatu data dengan melihat "k" tetangga terdekatnya dan memilih kelas yang paling sering muncul di antara tetangga tersebut. Sedangkan untuk regresi, prediksi nilai output diambil dari rata-rata nilai tetangga terdekat. KNN tidak memerlukan model eksplisit dan bekerja dengan baik pada data yang tidak terlalu besar serta memiliki distribusi yang jelas. Namun, algoritma ini bisa lambat saat data besar karena memerlukan perhitungan jarak antara titik data saat prediksi.

3. Multilayer Perceptron

Multilayer Perceptron (MLP) adalah jenis jaringan saraf tiruan yang terdiri dari beberapa lapisan neuron: lapisan input, satu atau lebih lapisan tersembunyi (hidden layers), dan lapisan output. MLP digunakan untuk tugas klasifikasi dan regresi. Setiap neuron dalam lapisan tersembunyi terhubung dengan neuron di lapisan sebelumnya dan selanjutnya, dengan bobot yang dioptimalkan selama proses pelatihan menggunakan algoritma seperti backpropagation. Proses ini memungkinkan MLP untuk belajar representasi kompleks dari data dan memetakan input ke output yang diinginkan. MLP sangat efektif dalam menangani masalah non-linear dan digunakan dalam berbagai aplikasi seperti pengenalan pola, prediksi, dan klasifikasi.

PEMBAHASAN

A. Code di Jupyter

1. K-Nearest Neighbor

```
[1]: import struct
import numpy as np
from sklearn import neighbors, metrics
import matplotlib.pyplot as plt
```

```
[3]: """
This function reads the MNIST data from the IDX file format into numpy arrays

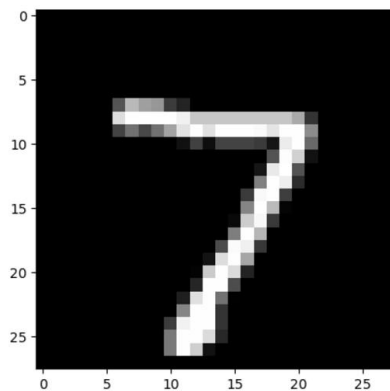
this relies on the fact that MNIST dataset consistently uses unsigned types
with heir data segments.
sc : https://gist.github.com/tylernelon/ce60e8a06e7506ac45788443f7269e40
"""

def read_idx(filename):
    with open(filename, 'rb') as f:
        zero, data_type, dims = struct.unpack('>HBB', f.read(4))
        shape = tuple(struct.unpack('>I', f.read(4))[0] for d in range(dims))
        return np.frombuffer(f.read(), dtype=np.uint8).reshape(shape)
```

```
[7]: raw_train = read_idx('./jupyter/train-images.idx3-ubyte')
train_data = np.reshape(raw_train, (60000, 28*28))
train_label = read_idx('./jupyter/train-labels.idx1-ubyte')

raw_test = read_idx('./jupyter/t10k-images.idx3-ubyte')
test_data = np.reshape(raw_test, (10000, 28*28))
test_label = read_idx('./jupyter/t10k-labels.idx1-ubyte')
```

```
[9]: plt.imshow(raw_test[0], cmap='gray')
plt.show()
```



```
[11]: X_train = train_data
Y_train = train_label
#K = 3
knn = neighbors.KNeighborsClassifier(n_neighbors=3).fit(X_train, Y_train)
```

```
[13]: X_test = test_data
Y_true = test_label
Y_pred = knn.predict(X_test)
```

```
[14]: con_matrix = metrics.confusion_matrix(Y_true, Y_pred)
```

```
[15]: from sklearn.model_selection import train_test_split
from sklearn.metrics import ConfusionMatrixDisplay
```

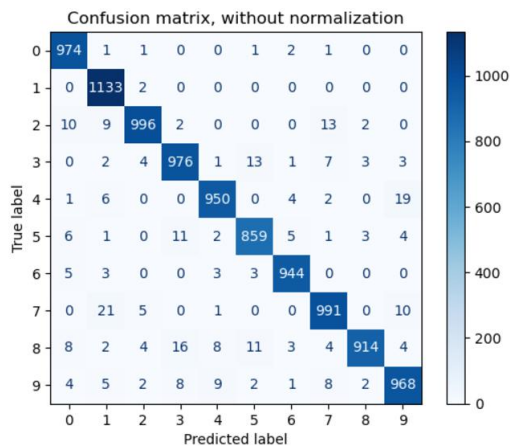
```
[16]: # Plot non-normalized confusion matrix
titles_options = [
    ("Confusion matrix, without normalization", None),
    ("Normalized confusion matrix", "true"),
]
for title, normalize in titles_options:
    values_format = ".2f" if normalize == "true" else None
    disp = ConfusionMatrixDisplay.from_predictions(
        y_true,
        y_pred,
        cmap=plt.cm.Blues,
        normalize=normalize,
        values_format=values_format
    )
    disp.ax_.set_title(title)

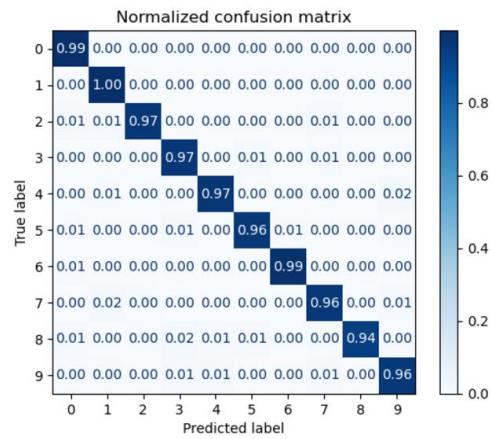
    print(title)
    print(disp.confusion_matrix)

plt.show()
```

```
Confusion matrix, without normalization
[[ 974  1  1  0  0  1  2  1  0  0]
 [  0 1133  2  0  0  0  0  0  0  0]
 [ 10  9 996  2  0  0  0 13  2  0]
 [  0  2  4 976  1 13  1  7  3  3]
 [  1  6  0  0 950  0  4  2  0 19]
 [  6  1  0 11  2 859  5  1  3  4]
 [  5  3  0  0  3  3 944  0  0  0]
 [  0 21  5  0  1  0  0 991  0 10]
 [  8  2  4 16  8 11  3  4 914  4]
 [  4  5  2  8  9  2  1  8  2 968]]
```

```
Normalized confusion matrix
[[9.93877551e-01 1.02040816e-03 1.02040816e-03 0.00000000e+00
 0.00000000e+00 1.02040816e-03 2.04081633e-03 1.02040816e-03
 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 9.98237885e-01 1.76211454e-03 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00]
 [9.68992248e-03 8.72093023e-03 9.65116279e-01 1.93798450e-03
 0.00000000e+00 0.00000000e+00 0.00000000e+00 1.25968992e-02
 1.93798450e-03 0.00000000e+00]
 [0.00000000e+00 1.98019802e-03 3.96039604e-03 9.66336634e-01
 9.90099010e-04 1.28712871e-02 9.90099010e-04 6.93069307e-03
 2.97029703e-03 2.97029703e-03]
 [1.01832994e-03 6.10997963e-03 0.00000000e+00 0.00000000e+00
 9.67413442e-01 0.00000000e+00 4.07331976e-03 2.03665988e-03
 0.00000000e+00 1.93482688e-02]
 [6.72645740e-03 1.12107623e-03 0.00000000e+00 1.23318386e-02
 2.24215247e-03 9.63004484e-01 5.60538117e-03 1.12107623e-03
 3.36322870e-03 4.48430493e-03]
 [5.21920668e-03 3.13152401e-03 0.00000000e+00 0.00000000e+00
 3.13152401e-03 3.13152401e-03 9.85386221e-01 0.00000000e+00
 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 2.04280156e-02 4.86381323e-03 0.00000000e+00
 9.72762646e-04 0.00000000e+00 0.00000000e+00 9.64007782e-01
 0.00000000e+00 9.72762646e-03]
 [8.21355236e-03 2.05338809e-03 4.10677618e-03 1.64271047e-02
 8.21355236e-03 1.12936345e-02 3.08008214e-03 4.10677618e-03
 9.38398357e-01 4.10677618e-03]
 [3.96432111e-03 4.95540139e-03 1.98216056e-03 7.92864222e-03
 8.91972250e-03 1.98216056e-03 9.91080278e-04 7.92864222e-03
 1.98216056e-03 9.59365709e-01]]
```



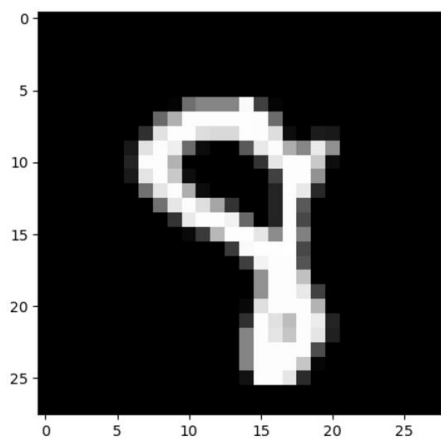
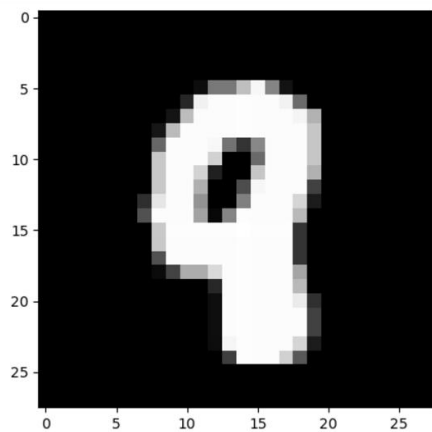


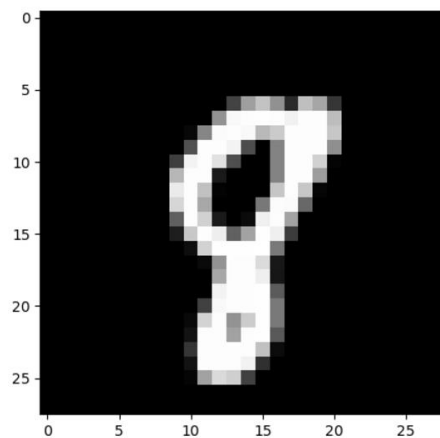
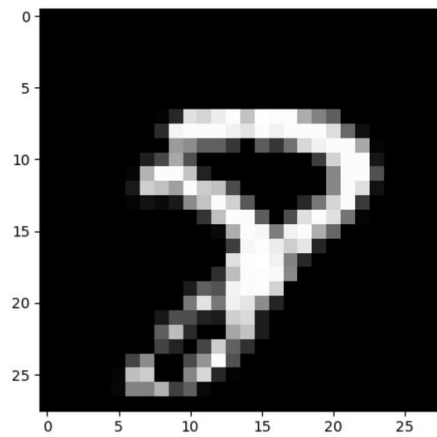
```
[21]: accuracy = metrics.accuracy_score(Y_true, Y_pred)
print(f'Accuracy: {accuracy}')
print(f'Accuracy %: {accuracy*100}%')
```

```
Accuracy: 0.9705
Accuracy %: 97.05%
```

```
[23]: idx = np.where((Y_pred == 9) & (Y_true == 8))[0]
fig = plt.figure(figsize=(5, 30))
for i in range(len(idx)):
    ax = fig.add_subplot(len(idx), 1, i+1)
    imgplot = ax.imshow(X_test[idx[i]].reshape(28, 28), cmap='gray')
    imgplot.set_interpolation('nearest')

plt.show()
```





```
[39]: #pick 2, 3, and 8

idx = (train_label == 2) | (train_label == 3) | (train_label == 8)

X_train = train_data[idx]
Y_train = train_label[idx]

knn = neighbors.KNeighborsClassifier(n_neighbors=3).fit(X_train, Y_train)
```

```
[41]: # test data for the classifier
idx = (test_label == 2) | (test_label == 3) | (test_label == 8)
X_test = test_data[idx]
Y_true = test_label[idx]
Y_pred = knn.predict(X_test)
```

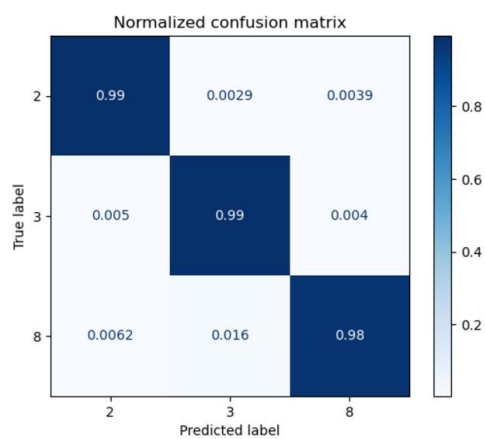
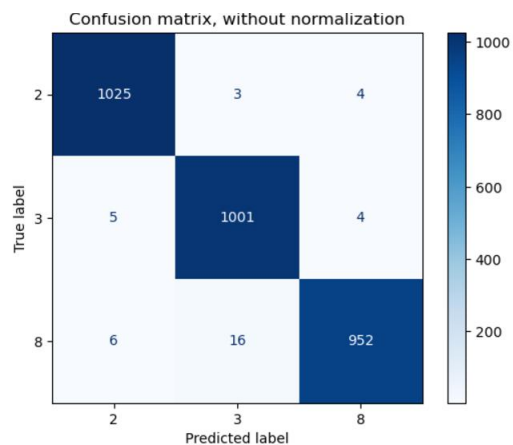
```
[43]: con_matrix = metrics.confusion_matrix(Y_true, Y_pred)
```

```
[45]: # Plot non-normalized confusion matrix
titles_options = [
    ("Confusion matrix, without normalization", None),
    ("Normalized confusion matrix", "true"),
]
for title, normalize in titles_options:
    disp = ConfusionMatrixDisplay.from_predictions(
        Y_true,
        Y_pred,
        cmap=plt.cm.Blues,
        normalize=normalize,
    )
    disp.ax_.set_title(title)

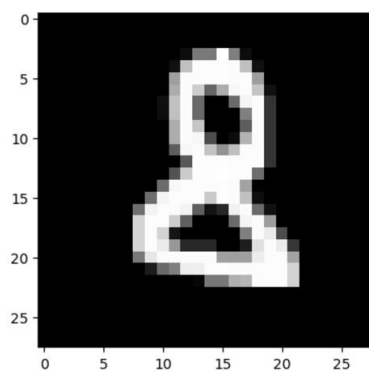
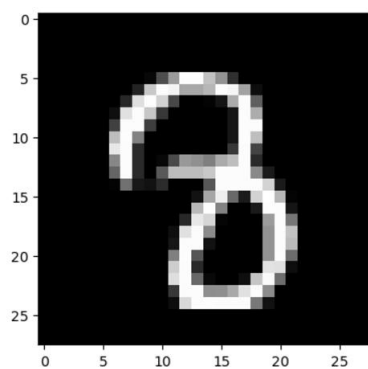
    print(title)
    print(disp.confusion_matrix)

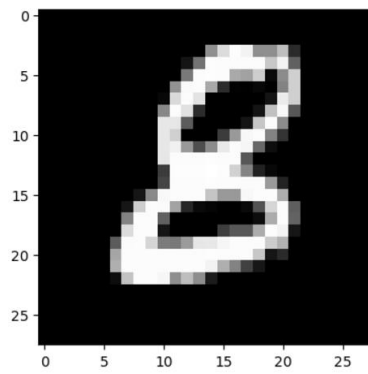
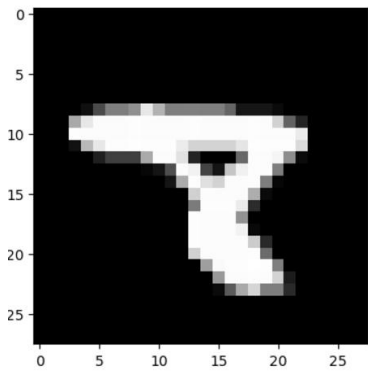
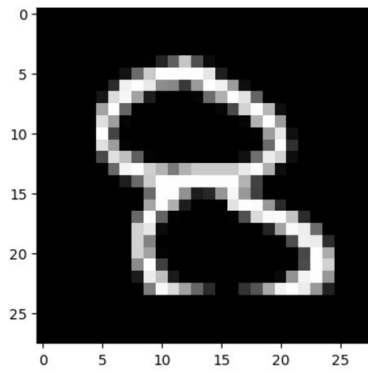
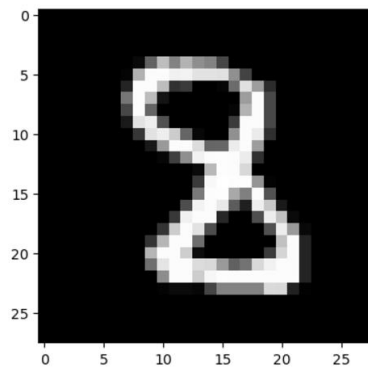
plt.show()
```

```
Confusion matrix, without normalization
[[1025   3   4]
 [   5 1001   4]
 [   6   16 952]]
Normalized confusion matrix
[[0.99321705 0.00290698 0.00387597]
 [0.0049505  0.99108911 0.0039604 ]
 [0.00616016 0.0164271  0.97741273]]
```



```
[47]: idx = np.where((Y_pred == 2) & (Y_true == 8))[0]
fig = plt.figure(figsize=(5, 30))
for i in range(len(idx)):
    ax = fig.add_subplot(len(idx), 1, i+1)
    imgplot = ax.imshow(X_test[idx[i]].reshape(28, 28), cmap='gray')
    imgplot.set_interpolation('nearest')
plt.show()
```





2. MLP

```
[1]: import struct
import numpy as np
import matplotlib.pyplot as plt

from sklearn import metrics
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import ConfusionMatrixDisplay

[3]: """
This function reads the MNIST data from the IDX file format into numpy arrays

this relies on the fact that MNIST dataset consistently uses unsigned types
with their data segments.
sc : https://gist.github.com/tylernelon/ce60e8a06e7506ac45788443f7269e40
"""

def read_idx(filename):
    with open(filename, 'rb') as f:
        zero, data_type, dims = struct.unpack('>HBB', f.read(4))
        shape = tuple(struct.unpack('>I', f.read(4))[0] for d in range(dims))
        return np.frombuffer(f.read(), dtype=np.uint8).reshape(shape)

[5]: raw_train = read_idx('./jupyter/train-images.idx3-ubyte')
train_data = np.reshape(raw_train, (60000, 28*28))
train_label = read_idx('./jupyter/train-labels.idx1-ubyte')

raw_test = read_idx('./jupyter/t10k-images.idx3-ubyte')
test_data = np.reshape(raw_test, (10000, 28*28))
test_label = read_idx('./jupyter/t10k-labels.idx1-ubyte')

[7]: X_train = train_data
Y_train = train_label
X_test = test_data
Y_true = test_label

# Initialize the MLPClassifier with increased max_iter
mlp = MLPClassifier(hidden_layer_sizes=500, max_iter=200, alpha=0.1)
# tuning reference : https://dmkothari.github.io/Machine-Learning-Projects/MLP\_with\_MNIST.html

# Train the MLPClassifier
mlp.fit(X_train, Y_train)

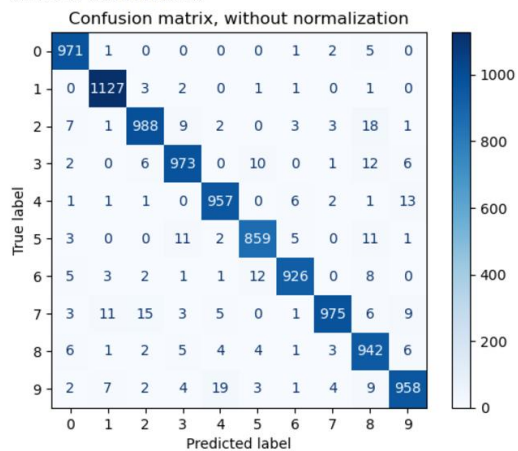
# Predict the Labels for the validation set
Y_pred = mlp.predict(X_test)

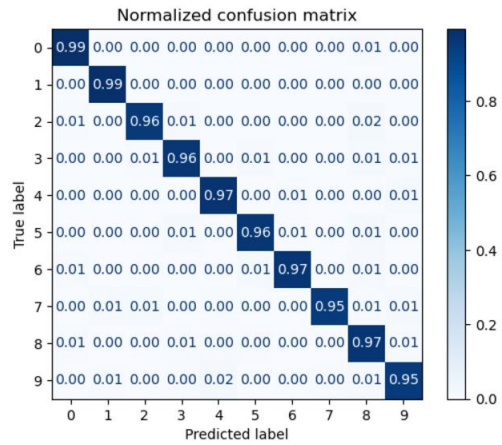
[9]: # Plot confusion matrix
titles_options = [
    ("Confusion matrix, without normalization", None),
    ("Normalized confusion matrix", "true"),
]
for title, normalize in titles_options:
    values_format = ".2f" if normalize == "true" else None
    disp = ConfusionMatrixDisplay.from_predictions(
        Y_true,
        Y_pred,
        cmap=plt.cm.Blues,
        normalize=normalize,
        values_format=values_format
    )
    disp.ax_.set_title(title)

    print(title)
    # print(disp.confusion_matrix)

plt.show()
```

Confusion matrix, without normalization
Normalized confusion matrix



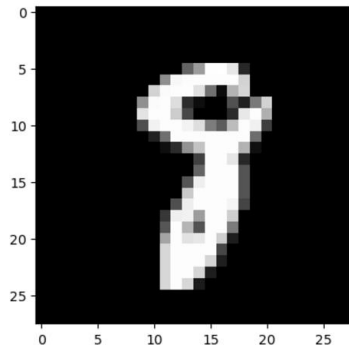


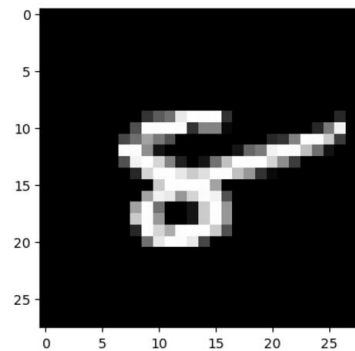
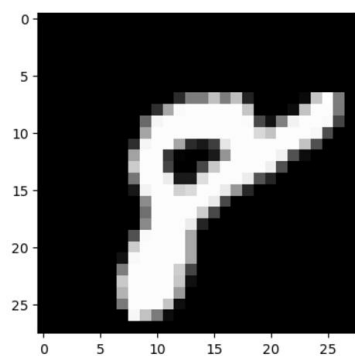
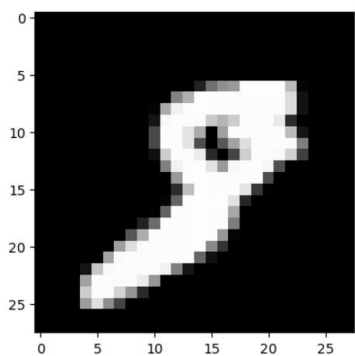
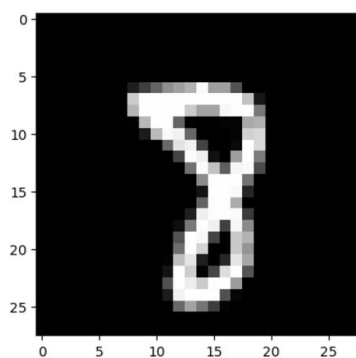
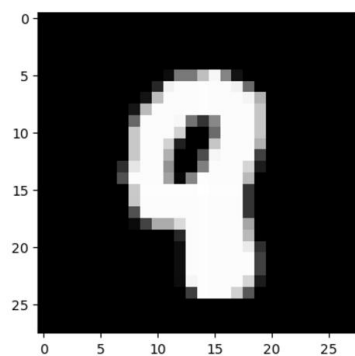
```
[12]: # Calculate the accuracy
accuracy = metrics.accuracy_score(Y_true, Y_pred)
print(f'Accuracy: {accuracy}')
print(f'Accuracy %: {accuracy*100}%')

Accuracy: 0.9676
Accuracy %: 96.76%
```

```
[14]: idx = np.where((Y_pred == 9) & (Y_true == 8))[0]
fig = plt.figure(figsize=(5, 30))
for i in range(len(idx)):
    ax = fig.add_subplot(len(idx), 1, i+1)
    imgplot = ax.imshow(X_test[idx[i]].reshape(28, 28), cmap='gray')
    imgplot.set_interpolation('nearest')

plt.show()
```





3. Naive Bayes

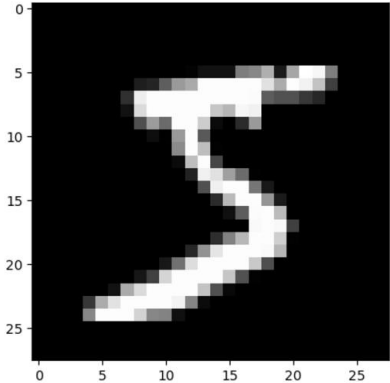
```
[7]: import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
from tensorflow.keras.datasets import mnist

[11]: (x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)

[12]: x_test

[12]: array([[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]], dtype=uint8)

[15]: plt.imshow(x_train[0].reshape(28, 28), cmap='gray')
plt.show()



[17]: nb_model = GaussianNB()

[19]: fit_nb = nb_model.fit(x_train, y_train)

[21]: predictions = fit_nb.predict(x_test)
con_matrix = confusion_matrix(y_test, predictions)
print(con_matrix)

[[ 870   0   3   5   2   5  31   1  35  28]
 [   0 1079   2   1   0   0  10   0  38   5]
 [  79  25 266  91   5   2 269   4 271  20]
 [  32  39   6 353   2   3  51   8 409 107]
 [  19   2   5   4 168   7  63   7 210 497]
 [  71 25   1  20   3  44  40   2 586 100]
 [  12 12   3   1   1   7 895   0  26   1]
 [   0 15   2 10   5   1   5 280  39 671]
 [  13 72   3   7   3  11  12   4 648 201]
 [   5   7   3   6   1   0   1  13  18 955]]

[23]: #con matrix plot
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import ConfusionMatrixDisplay

[25]: accuracy = metrics.accuracy_score(y_test, predictions)
print(f'Accuracy: {accuracy}')
print(f'Accuracy %: {accuracy*100}%')

Accuracy: 0.5558
Accuracy %: 55.58%

[27]: def diagonal_sum(con_matrix):
    sum = 0
    for i in range(10):
        for j in range(10):
            if i == j:
                sum += con_matrix[i][j]
    return sum
```

```
[29]: sum = diagonal_sum(con_matrix)
print(sum)
print(f'Accuracy %: {sum/100}')

5558
Accuracy %: 55.58

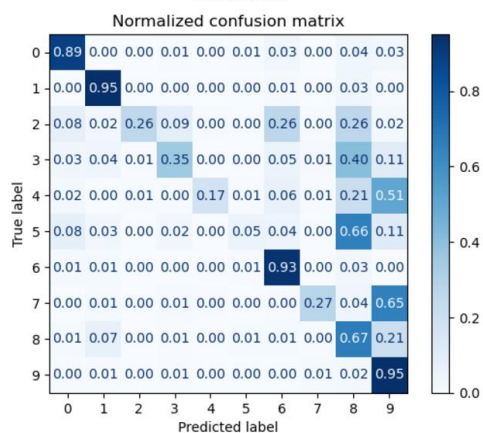
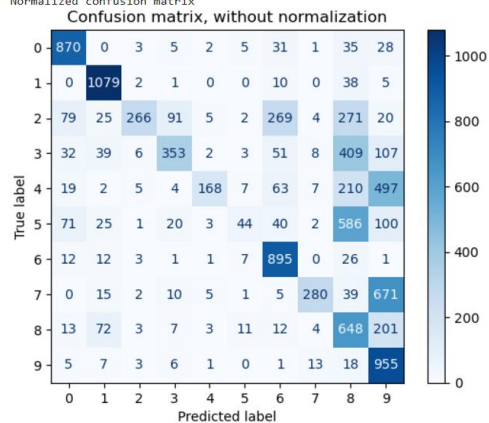
[31]: titles_options = [
    ("Confusion matrix, without normalization", None),
    ("Normalized confusion matrix", "true"),
]
for title, normalize in titles_options:
    values_format = ".2f" if normalize == "true" else None
    disp = ConfusionMatrixDisplay.from_predictions(
        y_test,
        predictions,
        cmap='Blues',
        normalize=normalize,
        values_format=values_format
    )
    disp.ax_.set_title(title)

    print(title)
    # print(disp.confusion_matrix)

plt.show()
```

Confusion matrix, without normalization

Normalized confusion matrix



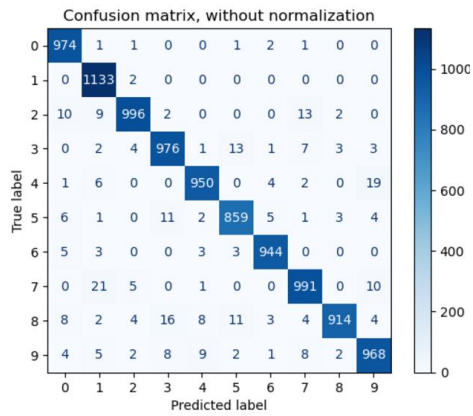
B. Perbandingan

Aspek untuk pembandingan dari KNN, MLP, dan Naive Bayes adalah tingkat akurasi nya

1. K-Nearest Neighbor

Accuracy: 0.9705

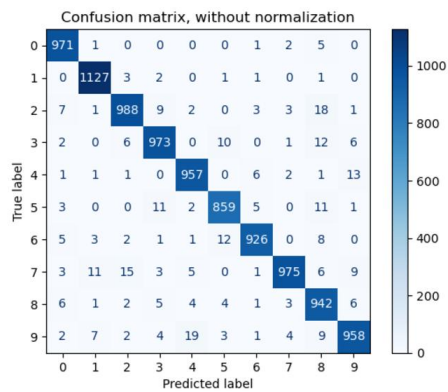
Accuracy %: 97.05%



2. MLP

Accuracy: 0.9676

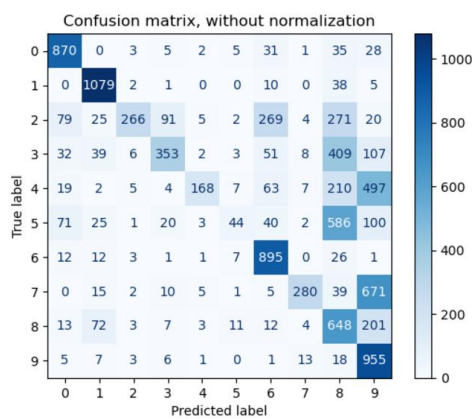
Accuracy %: 96.76%



3. Naive Bayes

Accuracy: 0.5558

Accuracy %: 55.58%



KESIMPULAN

Dari ketiga metode yang diberikan, KNN menjadi yang paling akurat dengan tingkat akurasi 97.05% disusul dengan MLP dengan tingkat akurasi 96.76%. Naive Bayes memiliki tingkat akurasi yang terendah dengan tingkat akurasi 55.58%. Berikut beberapa alasan singkat mengapa bisa terjadi.

1. Naive Bayes (55.58%)

Memiliki akurasi terendah karena membuat asumsi independensi antara fitur-fitur dalam data, yang sering kali tidak sesuai dengan kenyataan, terutama jika fitur-fitur tersebut saling bergantung. Hal ini dapat menyebabkan performa yang kurang baik, terutama pada dataset yang lebih kompleks atau tidak terstruktur.

2. Multilayer Perceptron, MLP (96.76%)

Memiliki akurasi tinggi karena menggunakan beberapa lapisan tersembunyi untuk belajar representasi non-linear yang lebih kompleks dari data. Kemampuan MLP untuk menangani hubungan non-linear antar fitur membuatnya sangat efektif pada masalah yang sulit dipisahkan secara linear.

3. K-Nearest Neighbors, KNN (97.05%)

Memiliki akurasi tinggi karena mengklasifikasikan data berdasarkan kedekatannya dengan data lain. KNN sangat baik untuk dataset yang memiliki pola yang jelas dan terstruktur, namun peformanya bisa menurun jika data sangat besar atau memiliki banyak fitur, karena perhitungan jarak antar data menjadi lebih mahal