In [27]:
```
import numpy as np
from typing import Tuple
import warnings
warnings.filterwarnings('ignore')

def load_data(filename: str) -> Tuple[np.ndarray, np.ndarray, np.ndarray]:
    data = np.load(f'{filename}')
    return data['features'], data['domains'], data['digits']

features_train, domains_train, digits_train = load_data('train_data.npz')
features_test, domains_test, digits_test = load_data('test_data.npz')

print(features_train.shape)
print(np.unique(domains_train, return_counts=True))

# print(features_test.shape)
# print(np.unique(domains_test, return_counts=True))
```

```
(25000, 1024)
(array([0, 1, 2, 3, 4], dtype=int64), array([5000, 5000, 5000, 5000, 5000], dtype=in
t64))
```

Explanation:

First of all, we cut data to having normal

In [28]:
```
import pandas as pd
from copy import deepcopy

features_train = pd.DataFrame(features_train)
features_train_copy = deepcopy(features_train)
domains_train = pd.DataFrame(domains_train)
features_test = pd.DataFrame(features_test)
domains_test = pd.DataFrame(domains_test)
digits_test = pd.DataFrame(digits_test)
digits_train = pd.DataFrame(digits_train)

train_0 = features_train.loc[domains_train[0] == 0]
features_train_0 = features_train.iloc[train_0.index]
features_train_0_copy = deepcopy(features_train_0)
features_train_0 = features_train_0[:500]
train_1 = features_train.loc[domains_train[0] == 1]
features_train_1 = features_train.iloc[train_1.index]
features_train_1_copy = deepcopy(features_train_1)
features_train_1 = features_train_1[:500]
train_2 = features_train.loc[domains_train[0] == 2]
features_train_2 = features_train.iloc[train_2.index]
features_train_2_copy = deepcopy(features_train_2)
features_train_2 = features_train_2[:500]
train_3 = features_train.loc[domains_train[0] == 3]
features_train_3 = features_train.iloc[train_3.index]
features_train_3_copy = deepcopy(features_train_3)
```

```
features_train_3 = features_train_3[:500]
train_4 = features_train.loc[domains_train[0] == 4]
features_train_4 = features_train.iloc[train_4.index]
features_train_4_copy = deepcopy(features_train_4)
features_train_4 = features_train_4[:500]
features_train = pd.concat([features_train_0, features_train_1, features_train_2, f
digits_train.iloc[features_train.index]
```

Out[28]:

	0
882	7
883	5
884	5
885	9
886	6
...	...
20495	3
20496	2
20497	5
20498	2
20499	5

2500 rows × 1 columns

part 1

In [10]:
```
from sklearn.ensemble import RandomForestClassifier

n_estimators = [i for i in range(100, 300, 40)]
max_depths = [i for i in range(10, 50, 8)]
estimated_n_estimator = None
estimated_max_depth = None
best_accuracy = 0
```

Explanation:

So now we find proper parameters based on part of 25000 data that we have.

In [6]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score

for n_estimator in n_estimators:
    for max_depth in max_depths:
```

```
        clf = RandomForestClassifier(max_depth=max_depth, n_estimators=n_estimator,
        clf.fit(features_train, digits_train.iloc[features_train.index])
        acc = accuracy_score(clf.predict(features_test), digits_test)
        print(f'With n_estimator={n_estimator} and max_depth={max_depth} we get acc
        cm = confusion_matrix(digits_test, clf.predict(features_test), labels=clf.c
        print(cm)
        if acc > best_accuracy:
            best_accuracy = acc
            estimated_n_estimator = n_estimator
            estimated_max_depth = max_depth

print(best_accuracy)
print(estimated_max_depth)
print(estimated_n_estimator)
```

```
With n_estimator=100 and max_depth=10 we get acc = 0.75036
[[1994  319  131   26   28   24   40   16   61   16]
 [  10 3067   88   10   65    3   10   59   18   11]
 [  16  168 2286   44   96    8   10   38   37   16]
 [   9  160  178 1721   52  157    9   21  139   32]
 [   8  305   70   11 1823    9   37    9   14   76]
 [  14  122   86  148   46 1786   24   18   63   35]
 [  68  219   61   42   89  215 1489    4  111   12]
 [   5  273  235   20   33   12    5 1689   19   82]
 [  64  178  126   97   38   85   46    8 1463   57]
 [  55  129  197   75  125   69   12   77   78 1441]]
With n_estimator=100 and max_depth=18 we get acc = 0.75292
[[2080  262  100   25   44   29   32   14   49   20]
 [   8 3031   89   18   74    8   12   68   25    8]
 [  27  164 2272   51   82   11   12   41   38   21]
 [  26  145  174 1749   43  150   15   30  106   40]
 [  20  307   68   10 1797   17   39   15   21   68]
 [  15   82   74  161   41 1804   36   16   60   53]
 [  83  157   53   47   93  209 1536    2  113   17]
 [  16  267  246   32   45   22    6 1648   20   71]
 [  62  159  121  109   49   99   52   12 1459   40]
 [  73  102  166   84  132   73    9   73   99 1447]]
With n_estimator=100 and max_depth=26 we get acc = 0.75276
[[2104  252  105   24   46   30   26    7   45   16]
 [  13 3054   86   15   63    7    7   65   21   10]
 [  19  181 2268   38   86   19   10   41   43   14]
 [  13  161  184 1707   46  174   20   25  105   43]
 [  16  308   77   11 1804    7   39   14   16   70]
 [  14   91   84  160   46 1805   41   10   44   47]
 [  87  168   68   35   99  200 1516    5  122   10]
 [  13  271  263   28   34   16    5 1627   25   91]
 [  71  146  120  101   52  107   41   10 1477   37]
 [  69   96  150   88  137   77   20   73   91 1457]]
With n_estimator=100 and max_depth=34 we get acc = 0.75276
[[2104  252  105   24   46   30   26    7   45   16]
 [  13 3054   86   15   63    7    7   65   21   10]
 [  19  181 2268   38   86   19   10   41   43   14]
 [  13  161  184 1707   46  174   20   25  105   43]
 [  16  308   77   11 1804    7   39   14   16   70]
 [  14   91   84  160   46 1805   41   10   44   47]
 [  87  168   68   35   99  200 1516    5  122   10]
 [  13  271  263   28   34   16    5 1627   25   91]
 [  71  146  120  101   52  107   41   10 1477   37]
 [  69   96  150   88  137   77   20   73   91 1457]]
With n_estimator=100 and max_depth=42 we get acc = 0.75276
[[2104  252  105   24   46   30   26    7   45   16]
 [  13 3054   86   15   63    7    7   65   21   10]
 [  19  181 2268   38   86   19   10   41   43   14]
 [  13  161  184 1707   46  174   20   25  105   43]
 [  16  308   77   11 1804    7   39   14   16   70]
 [  14   91   84  160   46 1805   41   10   44   47]
 [  87  168   68   35   99  200 1516    5  122   10]
 [  13  271  263   28   34   16    5 1627   25   91]
 [  71  146  120  101   52  107   41   10 1477   37]
 [  69   96  150   88  137   77   20   73   91 1457]]
With n_estimator=140 and max_depth=10 we get acc = 0.75636
```

```
[[2020  309  137   25   23   21   33   17   57   13]
 [   7 3074   94   11   61    5    8   58   16    7]
 [  12  187 2295   45   84    6    7   36   31   16]
 [  10  190  169 1744   48  140    9   18  115   35]
 [  11  303   57    6 1855    8   34    9   14   65]
 [  11  119   87  133   50 1798   28   18   62   36]
 [  61  217   56   41   95  215 1497    2  116   10]
 [   2  281  246   19   35    9    7 1686   20   68]
 [  51  191  123   93   45   68   38    5 1502   46]
 [  59  132  186   76  133   70   12   76   76 1438]]
With n_estimator=140 and max_depth=18 we get acc = 0.76212
[[2089  269  100   27   43   27   28   14   46   12]
 [   5 3061   93   15   58    7    7   68   22    5]
 [  23  164 2306   45   80   10   10   34   33   14]
 [  23  154  160 1771   42  163    9   31   95   30]
 [  25  302   61    8 1826   12   31   13   20   64]
 [  13   86   68  154   43 1848   24   16   52   38]
 [  81  176   44   44   91  226 1536    1   96   15]
 [  15  275  243   27   39   15    7 1673   21   58]
 [  54  160  130  112   46   96   44   11 1479   30]
 [  72  105  160   90  133   74    5   76   79 1464]]
With n_estimator=140 and max_depth=26 we get acc = 0.76148
[[2093  271  122   18   39   32   23   11   37    9]
 [   4 3085   77   14   54    5    8   66   20    8]
 [  17  170 2312   42   68   13   12   35   34   16]
 [  10  168  160 1755   40  171   15   20  100   39]
 [  11  310   63    7 1821    8   40   12   16   74]
 [  14  100   80  147   45 1827   37    9   47   36]
 [  91  183   61   30   94  203 1535    2  102    9]
 [  13  265  254   33   33   13    4 1671   17   70]
 [  57  154  121   96   50  101   46   12 1497   28]
 [  69  108  153   97  142   81   10   77   80 1441]]
With n_estimator=140 and max_depth=34 we get acc = 0.76148
[[2093  271  122   18   39   32   23   11   37    9]
 [   4 3085   77   14   54    5    8   66   20    8]
 [  17  170 2312   42   68   13   12   35   34   16]
 [  10  168  160 1755   40  171   15   20  100   39]
 [  11  310   63    7 1821    8   40   12   16   74]
 [  14  100   80  147   45 1827   37    9   47   36]
 [  91  183   61   30   94  203 1535    2  102    9]
 [  13  265  254   33   33   13    4 1671   17   70]
 [  57  154  121   96   50  101   46   12 1497   28]
 [  69  108  153   97  142   81   10   77   80 1441]]
With n_estimator=140 and max_depth=42 we get acc = 0.76148
[[2093  271  122   18   39   32   23   11   37    9]
 [   4 3085   77   14   54    5    8   66   20    8]
 [  17  170 2312   42   68   13   12   35   34   16]
 [  10  168  160 1755   40  171   15   20  100   39]
 [  11  310   63    7 1821    8   40   12   16   74]
 [  14  100   80  147   45 1827   37    9   47   36]
 [  91  183   61   30   94  203 1535    2  102    9]
 [  13  265  254   33   33   13    4 1671   17   70]
 [  57  154  121   96   50  101   46   12 1497   28]
 [  69  108  153   97  142   81   10   77   80 1441]]
With n_estimator=180 and max_depth=10 we get acc = 0.7606
[[2027  306  136   25   25   20   35   14   59    8]
```

```
 [   7 3080   88   11   63    6    7   56   17    6]
 [  13  186 2312   42   77    8    7   30   30   14]
 [  13  189  166 1740   42  153    5   18  123   29]
 [  13  316   52    8 1847    6   33    9   10   68]
 [  11  122   77  120   41 1837   20   15   66   33]
 [  67  214   49   28   89  224 1512    0  116   11]
 [   3  285  248   12   23    8    8 1698   18   70]
 [  46  196  120   93   36   69   36    7 1510   49]
 [  64  134  181   75  128   67   11   66   80 1452]]
With n_estimator=180 and max_depth=18 we get acc = 0.76656
[[2093  262  110   23   44   27   27   13   47    9]
 [   5 3069   85   14   61   11    9   62   16    9]
 [  22  161 2323   49   73   10    9   30   30   12]
 [  19  154  166 1779   39  160    8   23   99   31]
 [  18  300   64    6 1834   10   30   14   17   69]
 [  14   84   63  144   43 1877   19   11   48   39]
 [  84  180   45   35   92  213 1529    2  120   10]
 [  12  272  239   19   36   19    2 1689   20   65]
 [  46  150  115  104   58   94   36   10 1514   35]
 [  77  106  166   92  126   72    6   77   79 1457]]
With n_estimator=180 and max_depth=26 we get acc = 0.76668
[[2107  267  119   17   39   28   22    6   41    9]
 [   5 3081   81   11   52    6   11   67   20    7]
 [  17  170 2324   39   73   12   10   30   33   11]
 [   8  171  161 1770   43  166   11   24   91   33]
 [  14  306   65    8 1832    3   33   11   14   76]
 [  10   95   74  130   36 1876   31   11   47   32]
 [  79  179   63   25   94  212 1539    3  106   10]
 [  11  278  234   24   31   16    4 1686   18   71]
 [  55  156  126  100   56  101   31   11 1503   23]
 [  69  119  144   92  138   89    9   77   72 1449]]
With n_estimator=180 and max_depth=34 we get acc = 0.76668
[[2107  267  119   17   39   28   22    6   41    9]
 [   5 3081   81   11   52    6   11   67   20    7]
 [  17  170 2324   39   73   12   10   30   33   11]
 [   8  171  161 1770   43  166   11   24   91   33]
 [  14  306   65    8 1832    3   33   11   14   76]
 [  10   95   74  130   36 1876   31   11   47   32]
 [  79  179   63   25   94  212 1539    3  106   10]
 [  11  278  234   24   31   16    4 1686   18   71]
 [  55  156  126  100   56  101   31   11 1503   23]
 [  69  119  144   92  138   89    9   77   72 1449]]
With n_estimator=180 and max_depth=42 we get acc = 0.76668
[[2107  267  119   17   39   28   22    6   41    9]
 [   5 3081   81   11   52    6   11   67   20    7]
 [  17  170 2324   39   73   12   10   30   33   11]
 [   8  171  161 1770   43  166   11   24   91   33]
 [  14  306   65    8 1832    3   33   11   14   76]
 [  10   95   74  130   36 1876   31   11   47   32]
 [  79  179   63   25   94  212 1539    3  106   10]
 [  11  278  234   24   31   16    4 1686   18   71]
 [  55  156  126  100   56  101   31   11 1503   23]
 [  69  119  144   92  138   89    9   77   72 1449]]
With n_estimator=220 and max_depth=10 we get acc = 0.76244
[[2025  315  143   19   23   21   30   13   55   11]
 [   6 3074   98    8   57   10    8   56   20    4]
```

```
 [  11  186 2328   34   68    7    6   36   27   16]
 [  13  189  168 1733   47  162    3   19  119   25]
 [  12  303   50    7 1868    7   29   11   11   64]
 [   9  116   78  114   39 1859   18    9   67   33]
 [  58  217   59   22   87  223 1516    0  117   11]
 [   3  286  249   10   22   12    7 1691   20   73]
 [  49  188  131   83   39   70   34    7 1522   39]
 [  67  128  178   75  137   74   10   68   76 1445]]
With n_estimator=220 and max_depth=18 we get acc = 0.7718
[[2105  267  105   18   36   27   27   11   47   12]
 [   6 3072   83   14   61   10    7   63   17    8]
 [  22  157 2322   43   80   11    8   31   30   15]
 [  18  154  162 1765   39  170    7   22  106   35]
 [  13  284   59    5 1858   10   35   11   18   69]
 [  16   82   66  119   46 1905   21    8   47   32]
 [  76  176   49   31   93  224 1551    2   99    9]
 [   9  271  235   22   30   19    5 1697   19   66]
 [  43  151  110   76   59   99   35    5 1547   37]
 [  75  104  152   89  131   78    5   67   84 1473]]
With n_estimator=220 and max_depth=26 we get acc = 0.77204
[[2108  261  122   17   38   28   25    8   37   11]
 [   6 3080   82   12   58    6   12   63   17    5]
 [  22  163 2331   35   72    9   10   35   29   13]
 [  13  169  156 1771   42  170   11   20   96   30]
 [  10  287   59    8 1856    7   35   11   12   77]
 [  12   87   63  118   43 1901   27   10   48   33]
 [  75  177   53   24   90  213 1568    1  100    9]
 [   9  274  233   19   24   12    5 1707   19   71]
 [  57  154  117   90   60  105   36   10 1509   24]
 [  92  115  147   85  130   79    8   67   65 1470]]
With n_estimator=220 and max_depth=34 we get acc = 0.77204
[[2108  261  122   17   38   28   25    8   37   11]
 [   6 3080   82   12   58    6   12   63   17    5]
 [  22  163 2331   35   72    9   10   35   29   13]
 [  13  169  156 1771   42  170   11   20   96   30]
 [  10  287   59    8 1856    7   35   11   12   77]
 [  12   87   63  118   43 1901   27   10   48   33]
 [  75  177   53   24   90  213 1568    1  100    9]
 [   9  274  233   19   24   12    5 1707   19   71]
 [  57  154  117   90   60  105   36   10 1509   24]
 [  92  115  147   85  130   79    8   67   65 1470]]
With n_estimator=220 and max_depth=42 we get acc = 0.77204
[[2108  261  122   17   38   28   25    8   37   11]
 [   6 3080   82   12   58    6   12   63   17    5]
 [  22  163 2331   35   72    9   10   35   29   13]
 [  13  169  156 1771   42  170   11   20   96   30]
 [  10  287   59    8 1856    7   35   11   12   77]
 [  12   87   63  118   43 1901   27   10   48   33]
 [  75  177   53   24   90  213 1568    1  100    9]
 [   9  274  233   19   24   12    5 1707   19   71]
 [  57  154  117   90   60  105   36   10 1509   24]
 [  92  115  147   85  130   79    8   67   65 1470]]
With n_estimator=260 and max_depth=10 we get acc = 0.76464
[[2018  321  138   19   22   24   31   14   58   10]
 [   5 3087   89    7   54    9    7   55   22    6]
 [  12  178 2346   36   68    4    6   33   24   12]
```

```
 [  13  197  169 1741   42  160    4   17  112   23]
 [  13  310   55    6 1855    7   30   11   12   63]
 [  11  117   78  108   40 1873   21   11   59   24]
 [  51  209   56   22   87  237 1521    1  114   12]
 [   4  284  247    6   19   14    5 1709   17   68]
 [  49  190  125   74   37   82   35    7 1526   37]
 [  62  134  196   70  134   78    9   66   69 1440]]
With n_estimator=260 and max_depth=18 we get acc = 0.77188
[[2100  267  107   17   38   26   26   12   52   10]
 [   5 3080   83   13   58    8    7   63   20    4]
 [  20  160 2326   44   77    8    7   31   32   14]
 [  21  162  169 1755   34  172    8   18  108   31]
 [  13  279   60    2 1872   10   32   11   15   68]
 [  16   86   67  108   47 1916   18    9   45   30]
 [  74  175   37   28   94  224 1553    2  112   11]
 [  10  273  228   18   34   18    4 1704   17   67]
 [  39  164  109   92   55   86   32    7 1537   41]
 [  83  107  162   79  126   86    3   73   85 1454]]
With n_estimator=260 and max_depth=26 we get acc = 0.77448
[[2114  266  115   18   38   26   24   10   32   12]
 [   4 3085   83   11   54    7   11   65   15    6]
 [  19  165 2332   30   78   11    9   34   31   10]
 [  15  175  147 1767   43  167    9   23  102   30]
 [  10  272   59    4 1877    5   36   11   14   74]
 [  11   89   66  111   39 1917   30    8   41   30]
 [  70  171   51   22   94  214 1576    3  105    4]
 [   9  271  237   18   27   13    4 1709   14   71]
 [  53  157  118   85   58   99   40    9 1521   22]
 [  93  110  158   85  136   69    8   65   70 1464]]
With n_estimator=260 and max_depth=34 we get acc = 0.77448
[[2114  266  115   18   38   26   24   10   32   12]
 [   4 3085   83   11   54    7   11   65   15    6]
 [  19  165 2332   30   78   11    9   34   31   10]
 [  15  175  147 1767   43  167    9   23  102   30]
 [  10  272   59    4 1877    5   36   11   14   74]
 [  11   89   66  111   39 1917   30    8   41   30]
 [  70  171   51   22   94  214 1576    3  105    4]
 [   9  271  237   18   27   13    4 1709   14   71]
 [  53  157  118   85   58   99   40    9 1521   22]
 [  93  110  158   85  136   69    8   65   70 1464]]
With n_estimator=260 and max_depth=42 we get acc = 0.77448
[[2114  266  115   18   38   26   24   10   32   12]
 [   4 3085   83   11   54    7   11   65   15    6]
 [  19  165 2332   30   78   11    9   34   31   10]
 [  15  175  147 1767   43  167    9   23  102   30]
 [  10  272   59    4 1877    5   36   11   14   74]
 [  11   89   66  111   39 1917   30    8   41   30]
 [  70  171   51   22   94  214 1576    3  105    4]
 [   9  271  237   18   27   13    4 1709   14   71]
 [  53  157  118   85   58   99   40    9 1521   22]
 [  93  110  158   85  136   69    8   65   70 1464]]
0.77448
26
260
```

Therefore best accuracy score is 77% with these parameters:

- number of trees: 260
- depth number: 26

part 2

as you see above, we divided our data into 5 partition:

train_0 = features_train.loc[domains_train[0] == 0] features_train_0 = features_train.iloc[train_0.index] features_train_0 = features_train_0[:500] train_1 = features_train.loc[domains_train[0] == 1] features_train_1 = features_train.iloc[train_1.index] features_train_1 = features_train_1[:500] train_2 = features_train.loc[domains_train[0] == 2] features_train_2 = features_train.iloc[train_2.index] features_train_2 = features_train_2[:500] train_3 = features_train.loc[domains_train[0] == 3] features_train_3 = features_train.iloc[train_3.index] features_train_3 = features_train_3[:500] train_4 = features_train.loc[domains_train[0] == 4] features_train_4 = features_train.iloc[train_4.index] features_train_4 = features_train_4[:500]

In [26]:
```
n_estimators = [i for i in range(100, 200, 25)]
max_depths = [i for i in range(5, 35, 10)]
estimated_n_estimator = None
estimated_max_depth = None
best_accuracy = 0

for n_estimator in n_estimators:
    for max_depth in max_depths:
        clf = RandomForestClassifier(max_depth=max_depth, n_estimators=n_estimator,
        clf.fit(features_train_0_copy[:2500], digits_train.iloc[features_train_0_co
        features_test_0 = features_test.loc[domains_test[0] == 0]
        digits_test_0 = digits_test.iloc[features_test_0.index]
        acc = accuracy_score(clf.predict(features_test_0), digits_test_0)
        print(f'With n_estimator={n_estimator} and max_depth={max_depth} we get acc
        if acc > best_accuracy:
            best_accuracy = acc
            estimated_n_estimator = n_estimator
            estimated_max_depth = max_depth

print(f'best_accuracy is {best_accuracy}')
print(f'estimated_max_depth is {estimated_max_depth}')
print(f'estimated_n_estimator is {estimated_n_estimator}')

# digits_train_0 = digits_train.iloc[domain_0.index]
# clf0.fit(features_domain_0.iloc[:2500], digits_train_0[:2500][0])
# features_test_0 = features_test.loc[domains_test[0] == 0]
# digits_test_0 = digits_test.iloc[features_test_0.index]
# predict_y_0 = clf0.predict(features_test_0)
# accuracy_score(digits_test_0, predict_y_0)
```

```
With n_estimator=100 and max_depth=5 we get acc = 0.9414
With n_estimator=100 and max_depth=15 we get acc = 0.9618
With n_estimator=100 and max_depth=25 we get acc = 0.9626
With n_estimator=125 and max_depth=5 we get acc = 0.9418
With n_estimator=125 and max_depth=15 we get acc = 0.964
With n_estimator=125 and max_depth=25 we get acc = 0.9642
With n_estimator=150 and max_depth=5 we get acc = 0.9422
With n_estimator=150 and max_depth=15 we get acc = 0.964
With n_estimator=150 and max_depth=25 we get acc = 0.963
With n_estimator=175 and max_depth=5 we get acc = 0.9434
With n_estimator=175 and max_depth=15 we get acc = 0.9644
With n_estimator=175 and max_depth=25 we get acc = 0.9636
best_accuracy is 0.9644
estimated_max_depth is 15
estimated_n_estimator is 175
```

In [29]:

```
n_estimators = [i for i in range(100, 200, 25)]
max_depths = [i for i in range(5, 35, 10)]
estimated_n_estimator = None
estimated_max_depth = None
best_accuracy = 0

for n_estimator in n_estimators:
    for max_depth in max_depths:
        clf = RandomForestClassifier(max_depth=max_depth, n_estimators=n_estimator,
        clf.fit(features_train_1_copy[:2500], digits_train.iloc[features_train_1_co
        features_test_1 = features_test.loc[domains_test[0] == 1]
        digits_test_1 = digits_test.iloc[features_test_1.index]
        acc = accuracy_score(clf.predict(features_test_1), digits_test_1)
        print(f'With n_estimator={n_estimator} and max_depth={max_depth} we get acc
        if acc > best_accuracy:
            best_accuracy = acc
            estimated_n_estimator = n_estimator
            estimated_max_depth = max_depth

print(f'best_accuracy is {best_accuracy}')
print(f'estimated_max_depth is {estimated_max_depth}')
print(f'estimated_n_estimator is {estimated_n_estimator}')


# clf1 = RandomForestClassifier(max_depth=3, random_state=0)
# digits_train_1 = digits_train.iloc[domain_1.index]
# clf1.fit(features_domain_1.iloc[:2500], digits_train_1[:2500][0])
# features_test_1 = features_test.loc[domains_test[0] == 1]
# digits_test_1 = digits_test.iloc[features_test_1.index]
# predict_y_1 = clf1.predict(features_test_1)
# accuracy_score(digits_test_1, predict_y_1)
```

```
With n_estimator=100 and max_depth=5 we get acc = 0.775
With n_estimator=100 and max_depth=15 we get acc = 0.8256
With n_estimator=100 and max_depth=25 we get acc = 0.8272
With n_estimator=125 and max_depth=5 we get acc = 0.7806
With n_estimator=125 and max_depth=15 we get acc = 0.8322
With n_estimator=125 and max_depth=25 we get acc = 0.8314
With n_estimator=150 and max_depth=5 we get acc = 0.7846
With n_estimator=150 and max_depth=15 we get acc = 0.835
With n_estimator=150 and max_depth=25 we get acc = 0.8336
With n_estimator=175 and max_depth=5 we get acc = 0.7876
With n_estimator=175 and max_depth=15 we get acc = 0.8368
With n_estimator=175 and max_depth=25 we get acc = 0.8426
best_accuracy is 0.8426
estimated_max_depth is 25
estimated_n_estimator is 175
```

In [30]:

```
n_estimators = [i for i in range(100, 200, 25)]
max_depths = [i for i in range(5, 35, 10)]
estimated_n_estimator = None
estimated_max_depth = None
best_accuracy = 0

for n_estimator in n_estimators:
    for max_depth in max_depths:
        clf = RandomForestClassifier(max_depth=max_depth, n_estimators=n_estimator,
        clf.fit(features_train_2_copy[:2500], digits_train.iloc[features_train_2_co
        features_test_2 = features_test.loc[domains_test[0] == 2]
        digits_test_2 = digits_test.iloc[features_test_2.index]
        acc = accuracy_score(clf.predict(features_test_2), digits_test_2)
        print(f'With n_estimator={n_estimator} and max_depth={max_depth} we get acc
        if acc > best_accuracy:
            best_accuracy = acc
            estimated_n_estimator = n_estimator
            estimated_max_depth = max_depth

print(f'best_accuracy is {best_accuracy}')
print(f'estimated_max_depth is {estimated_max_depth}')
print(f'estimated_n_estimator is {estimated_n_estimator}')


# clf2 = RandomForestClassifier(max_depth=3, random_state=0)
# digits_train_2 = digits_train.iloc[domain_2.index]
# clf2.fit(features_domain_2.iloc[:2500], digits_train_2[:2500][0])
# features_test_2 = features_test.loc[domains_test[0] == 2]
# digits_test_2 = digits_test.iloc[features_test_2.index]
# predict_y_2 = clf2.predict(features_test_2)
# accuracy_score(digits_test_2, predict_y_2)
```

```
With n_estimator=100 and max_depth=5 we get acc = 0.5878
With n_estimator=100 and max_depth=15 we get acc = 0.7084
With n_estimator=100 and max_depth=25 we get acc = 0.7062
With n_estimator=125 and max_depth=5 we get acc = 0.588
With n_estimator=125 and max_depth=15 we get acc = 0.714
With n_estimator=125 and max_depth=25 we get acc = 0.7116
With n_estimator=150 and max_depth=5 we get acc = 0.5908
With n_estimator=150 and max_depth=15 we get acc = 0.7196
With n_estimator=150 and max_depth=25 we get acc = 0.7168
With n_estimator=175 and max_depth=5 we get acc = 0.5902
With n_estimator=175 and max_depth=15 we get acc = 0.7228
With n_estimator=175 and max_depth=25 we get acc = 0.7212
best_accuracy is 0.7228
estimated_max_depth is 15
estimated_n_estimator is 175
```

In [31]:

```
n_estimators = [i for i in range(100, 200, 25)]
max_depths = [i for i in range(5, 35, 10)]
estimated_n_estimator = None
estimated_max_depth = None
best_accuracy = 0

for n_estimator in n_estimators:
    for max_depth in max_depths:
        clf = RandomForestClassifier(max_depth=max_depth, n_estimators=n_estimator,
        clf.fit(features_train_3_copy[:2500], digits_train.iloc[features_train_3_co
        features_test_3 = features_test.loc[domains_test[0] == 3]
        digits_test_3 = digits_test.iloc[features_test_3.index]
        acc = accuracy_score(clf.predict(features_test_3), digits_test_3)
        print(f'With n_estimator={n_estimator} and max_depth={max_depth} we get acc
        if acc > best_accuracy:
            best_accuracy = acc
            estimated_n_estimator = n_estimator
            estimated_max_depth = max_depth

print(f'best_accuracy is {best_accuracy}')
print(f'estimated_max_depth is {estimated_max_depth}')
print(f'estimated_n_estimator is {estimated_n_estimator}')


# clf3 = RandomForestClassifier(max_depth=3, random_state=0)
# digits_train_3 = digits_train.iloc[domain_3.index]
# clf3.fit(features_domain_3.iloc[:2500], digits_train_3[:2500][0])
# features_test_3 = features_test.loc[domains_test[0] == 3]
# digits_test_3 = digits_test.iloc[features_test_3.index]
# predict_y_3 = clf3.predict(features_test_3)
# accuracy_score(digits_test_3, predict_y_3)
```

```
With n_estimator=100 and max_depth=5 we get acc = 0.551
With n_estimator=100 and max_depth=15 we get acc = 0.6026
With n_estimator=100 and max_depth=25 we get acc = 0.5904
With n_estimator=125 and max_depth=5 we get acc = 0.5508
With n_estimator=125 and max_depth=15 we get acc = 0.6056
With n_estimator=125 and max_depth=25 we get acc = 0.5966
With n_estimator=150 and max_depth=5 we get acc = 0.5586
With n_estimator=150 and max_depth=15 we get acc = 0.6112
With n_estimator=150 and max_depth=25 we get acc = 0.6066
With n_estimator=175 and max_depth=5 we get acc = 0.5566
With n_estimator=175 and max_depth=15 we get acc = 0.616
With n_estimator=175 and max_depth=25 we get acc = 0.6122
best_accuracy is 0.616
estimated_max_depth is 15
estimated_n_estimator is 175
```

In [32]:

```
n_estimators = [i for i in range(100, 200, 25)]
max_depths = [i for i in range(5, 35, 10)]
estimated_n_estimator = None
estimated_max_depth = None
best_accuracy = 0

for n_estimator in n_estimators:
    for max_depth in max_depths:
        clf = RandomForestClassifier(max_depth=max_depth, n_estimators=n_estimator,
        clf.fit(features_train_4_copy[:2500], digits_train.iloc[features_train_4_co
        features_test_4 = features_test.loc[domains_test[0] == 4]
        digits_test_4 = digits_test.iloc[features_test_4.index]
        acc = accuracy_score(clf.predict(features_test_4), digits_test_4)
        print(f'With n_estimator={n_estimator} and max_depth={max_depth} we get acc
        if acc > best_accuracy:
            best_accuracy = acc
            estimated_n_estimator = n_estimator
            estimated_max_depth = max_depth

print(f'best_accuracy is {best_accuracy}')
print(f'estimated_max_depth is {estimated_max_depth}')
print(f'estimated_n_estimator is {estimated_n_estimator}')


# clf4 = RandomForestClassifier(max_depth=3, random_state=0)
# digits_train_4 = digits_train.iloc[domain_4.index]
# clf4.fit(features_domain_4.iloc[:2500], digits_train_4[:2500][0])
# features_test_4 = features_test.loc[domains_test[0] == 4]
# digits_test_4 = digits_test.iloc[features_test_4.index]
# predict_y_4 = clf4.predict(features_test_4)
# accuracy_score(digits_test_4, predict_y_4)
```

```
With n_estimator=100 and max_depth=5 we get acc = 0.9484
With n_estimator=100 and max_depth=15 we get acc = 0.974
With n_estimator=100 and max_depth=25 we get acc = 0.9732
With n_estimator=125 and max_depth=5 we get acc = 0.9486
With n_estimator=125 and max_depth=15 we get acc = 0.973
With n_estimator=125 and max_depth=25 we get acc = 0.9736
With n_estimator=150 and max_depth=5 we get acc = 0.9488
With n_estimator=150 and max_depth=15 we get acc = 0.9742
With n_estimator=150 and max_depth=25 we get acc = 0.9738
With n_estimator=175 and max_depth=5 we get acc = 0.9474
With n_estimator=175 and max_depth=15 we get acc = 0.9738
With n_estimator=175 and max_depth=25 we get acc = 0.9744
best_accuracy is 0.9744
estimated_max_depth is 25
estimated_n_estimator is 175
```

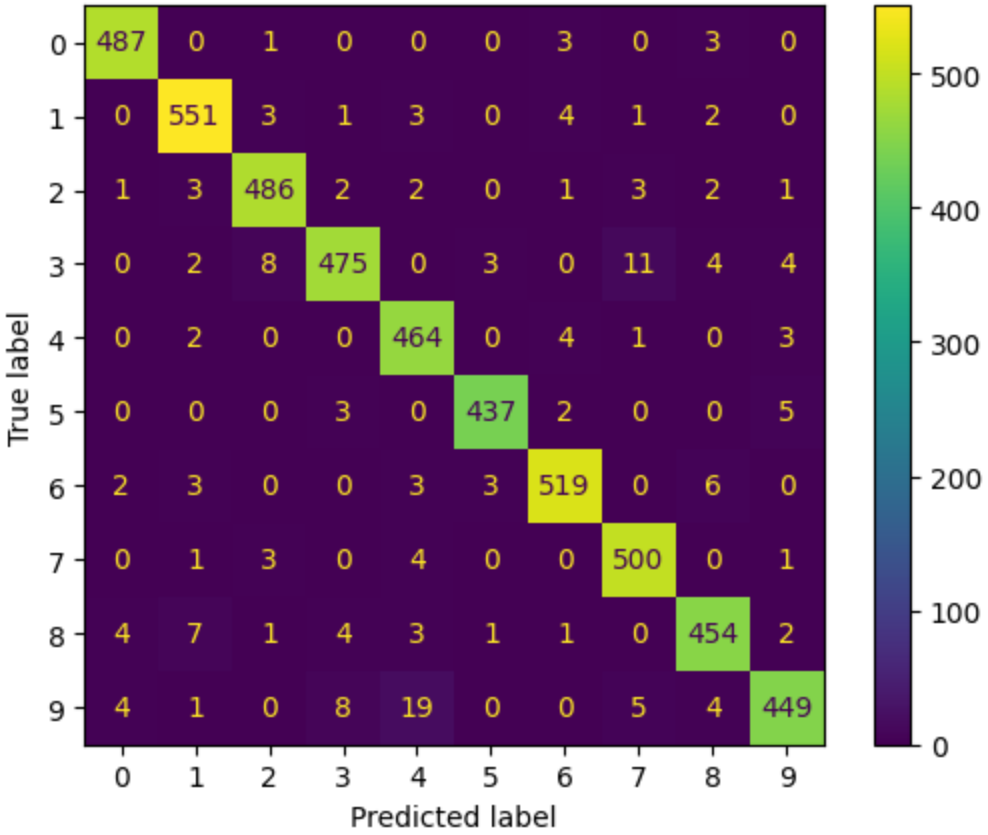
part 3

In [37]:
```
digits_train_0 = digits_train.iloc[features_train_0_copy.iloc[:2500].index]
digits_train_1 = digits_train.iloc[features_train_1_copy.iloc[:2500].index]
digits_train_2 = digits_train.iloc[features_train_2_copy.iloc[:2500].index]
digits_train_3 = digits_train.iloc[features_train_3_copy.iloc[:2500].index]
digits_train_4 = digits_train.iloc[features_train_4_copy.iloc[:2500].index]
```

Domain 0

previous accuracy was 96.44%

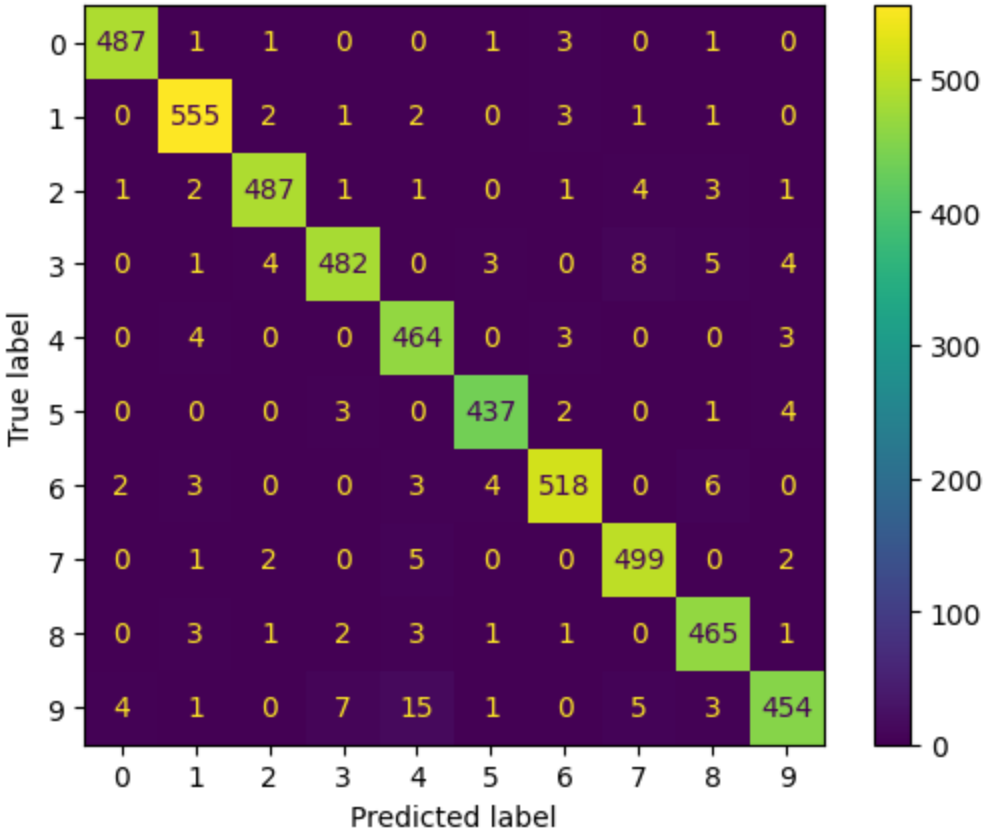
In [33]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

clf = RandomForestClassifier(max_depth=15, n_estimators=175, random_state=42)
clf.fit(features_train_0_copy[:2500], digits_train.iloc[features_train_0_copy.index
cm = confusion_matrix(digits_test_0, clf.predict(features_test_0), labels=clf.class
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
disp.plot()
plt.show()
```



In [38]:
```
features_0 = [features_train_0_copy.iloc[:2500], features_train_1_copy[:250], \
                        features_train_2_copy[:200], features_train_3_copy[:350], f
new_features_domain_0 = pd.concat(features_0)
to_append = [digits_train_0[:2500][0], digits_train_1[:250][0], digits_train_2[:200
new_digits_train_0 = pd.concat(to_append)
clf.fit(new_features_domain_0, new_digits_train_0)
new_predict_y_0 = clf.predict(features_test_0)
accuracy_score(digits_test_0, new_predict_y_0)
```

Out[38]: 0.9696

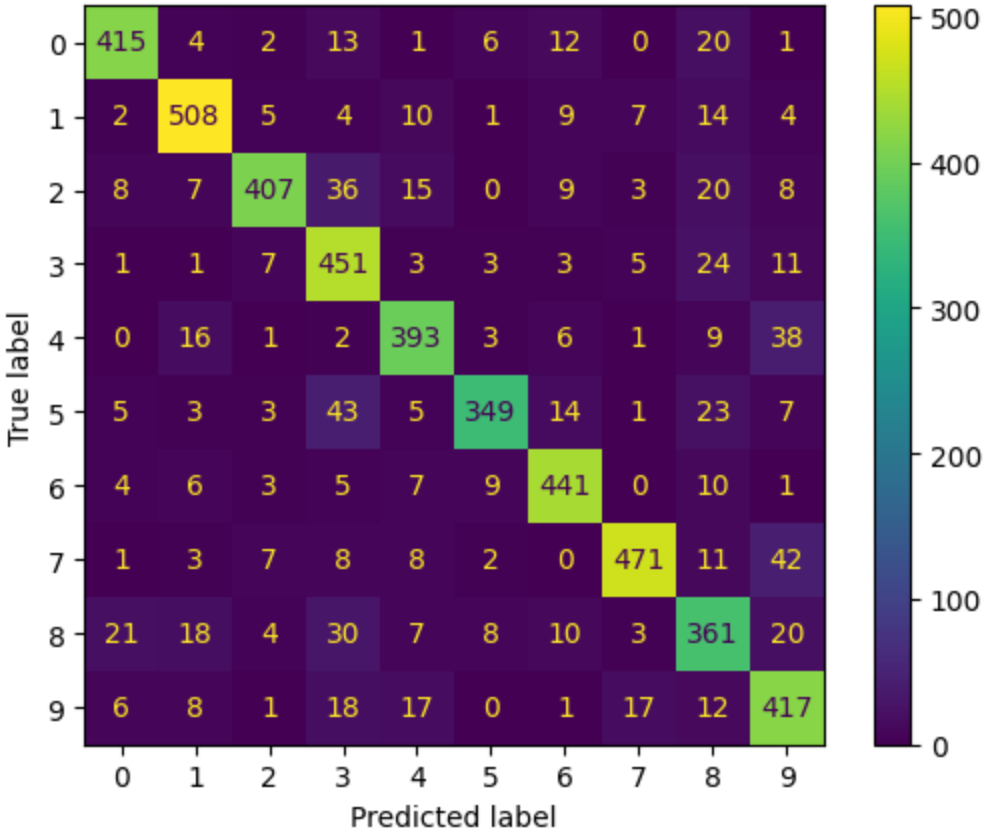
In [36]:
```
cm = confusion_matrix(digits_test_0, new_predict_y_0, labels=clf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
disp.plot()
plt.show()
```



Domain 1

previous accuracy was 84.26%

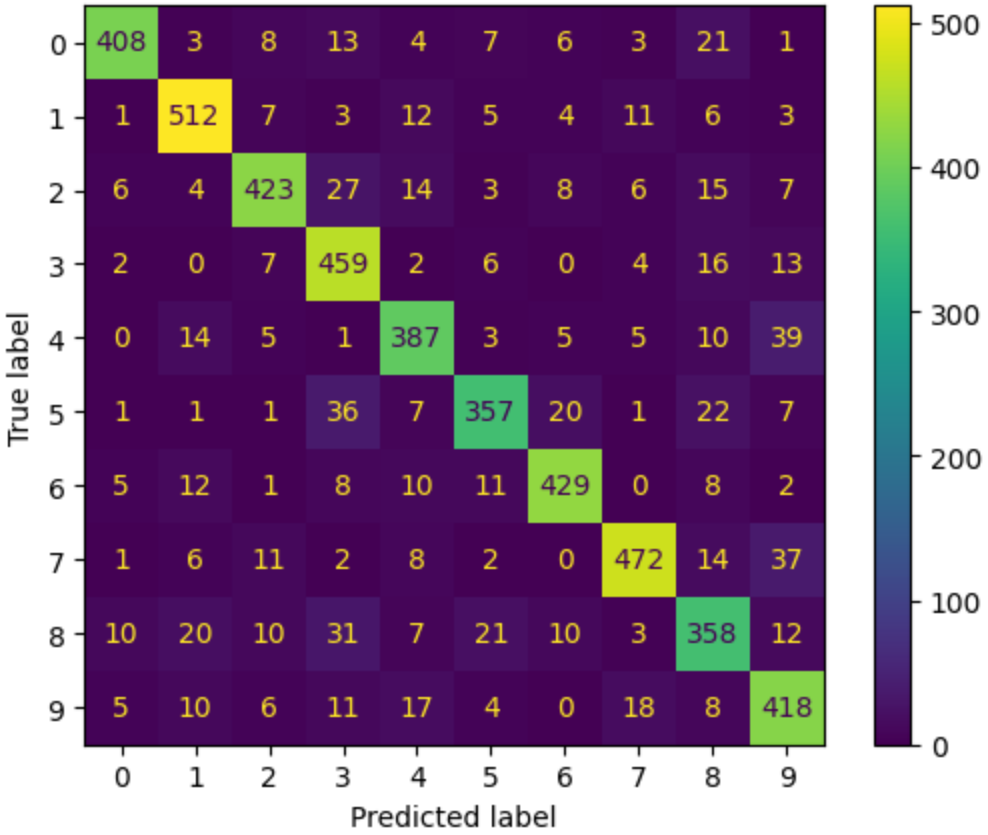
In [43]:
```
clf = RandomForestClassifier(max_depth=25, n_estimators=175, random_state=42)
clf.fit(features_train_1_copy[:2500], digits_train.iloc[features_train_1_copy.index
cm = confusion_matrix(digits_test_1, clf.predict(features_test_1), labels=clf.class
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
disp.plot()
plt.show()
```



In [44]:
```
features_1 = [features_train_1_copy.iloc[:2500], features_train_0_copy[:150], \
                        features_train_2_copy[:350], features_train_3_copy[:200], f
new_features_domain_1 = pd.concat(features_1)
to_append = [digits_train_1[:2500][0], digits_train_0[:150][0], digits_train_2[:350
new_digits_train_1 = pd.concat(to_append)
clf.fit(new_features_domain_1, new_digits_train_1)
new_predict_y_1 = clf.predict(features_test_1)
accuracy_score(digits_test_1, new_predict_y_1)
```

Out[44]: 0.8446

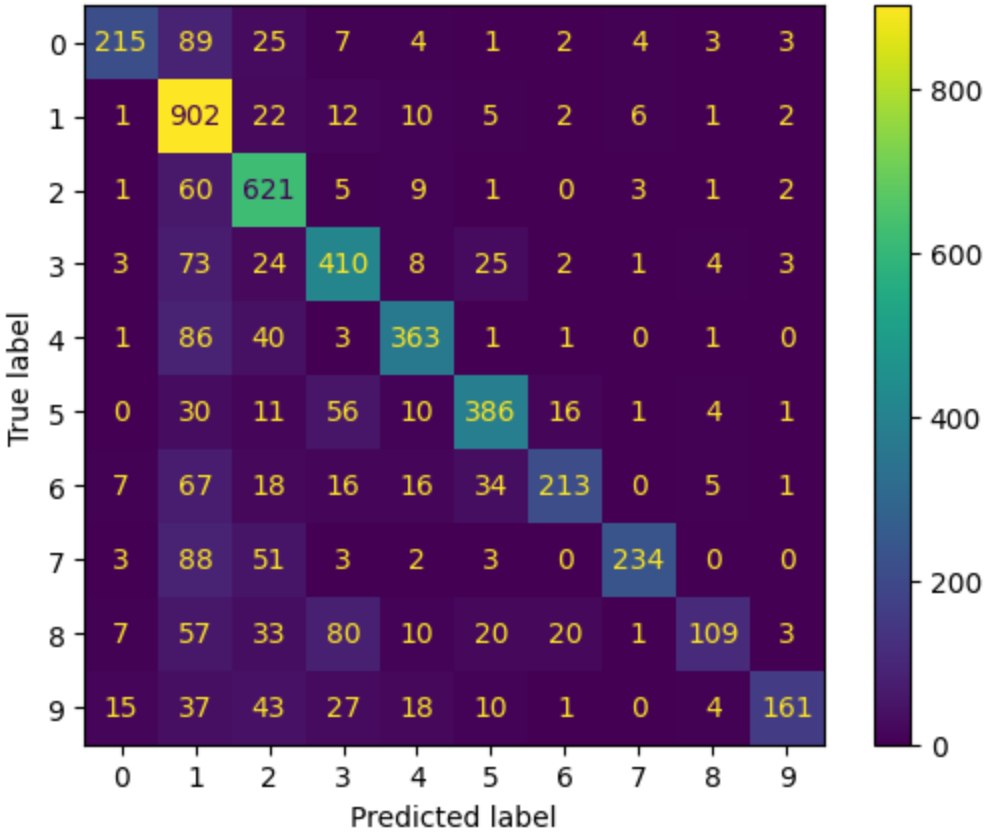
In [45]:
```
cm = confusion_matrix(digits_test_1, new_predict_y_1, labels=clf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
disp.plot()
plt.show()
```



Domain 2

previous accuracy was 72.28%

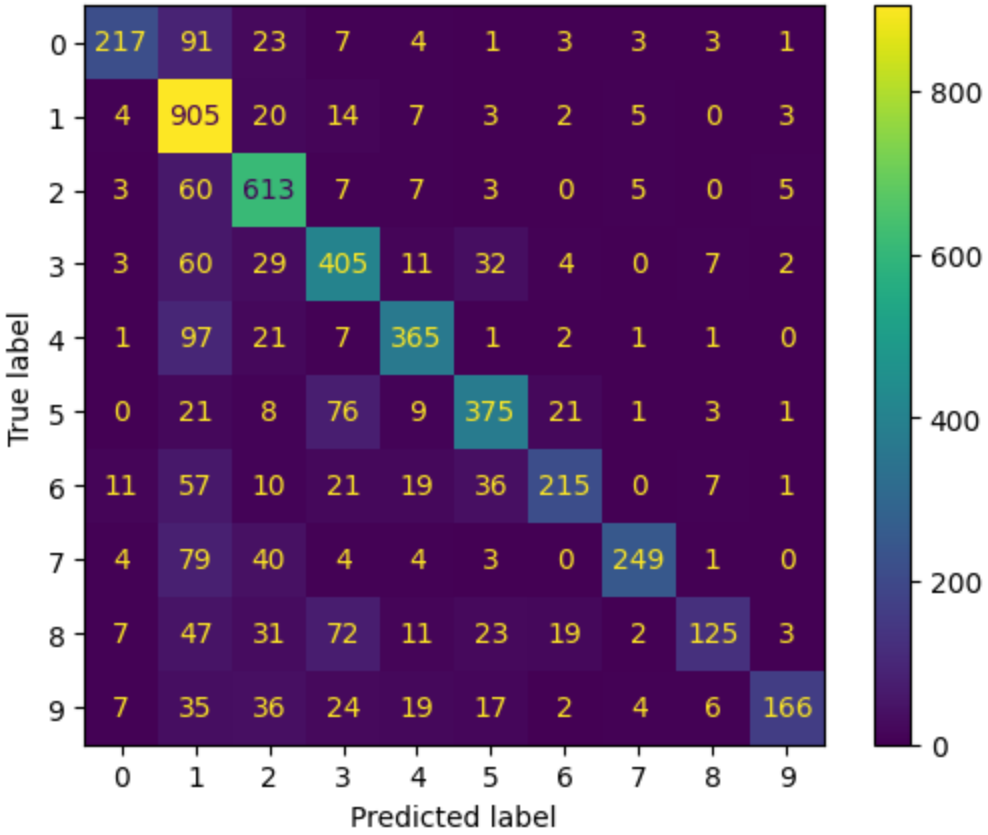
In [49]:
```
clf = RandomForestClassifier(max_depth=15, n_estimators=175, random_state=42)
clf.fit(features_train_2_copy[:2500], digits_train.iloc[features_train_2_copy.index
cm = confusion_matrix(digits_test_2, clf.predict(features_test_2), labels=clf.class
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
disp.plot()
plt.show()
```



In [53]:
```
features_2 = [features_train_2_copy.iloc[:2500], features_train_1_copy[:120], \
                        features_train_0_copy[:300], features_train_3_copy[:50], fe
new_features_domain_2 = pd.concat(features_2)
to_append = [digits_train_2[:2500][0], digits_train_1[:120][0], digits_train_0[:300
new_digits_train_2 = pd.concat(to_append)
clf.fit(new_features_domain_2, new_digits_train_2)
new_predict_y_2 = clf.predict(features_test_2)
accuracy_score(digits_test_2, new_predict_y_2)
```

Out[53]: 0.727

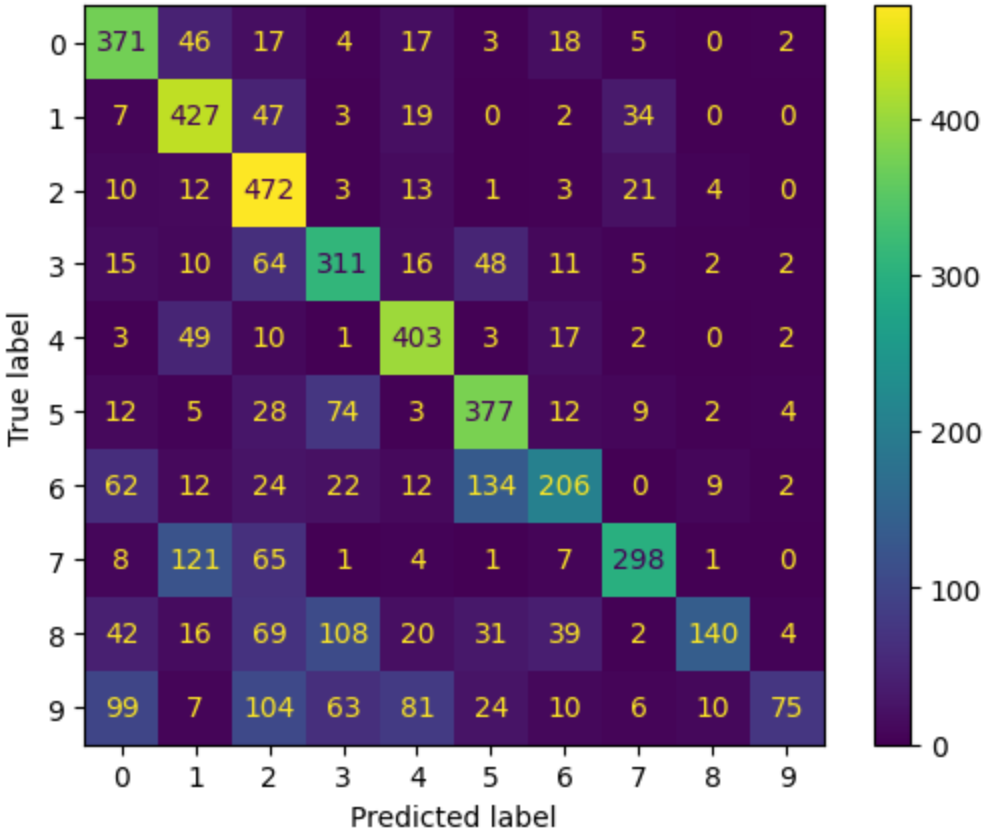
In [54]:
```
cm = confusion_matrix(digits_test_2, new_predict_y_2, labels=clf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
disp.plot()
plt.show()
```



Domain 3

previous accuracy was 61.6%

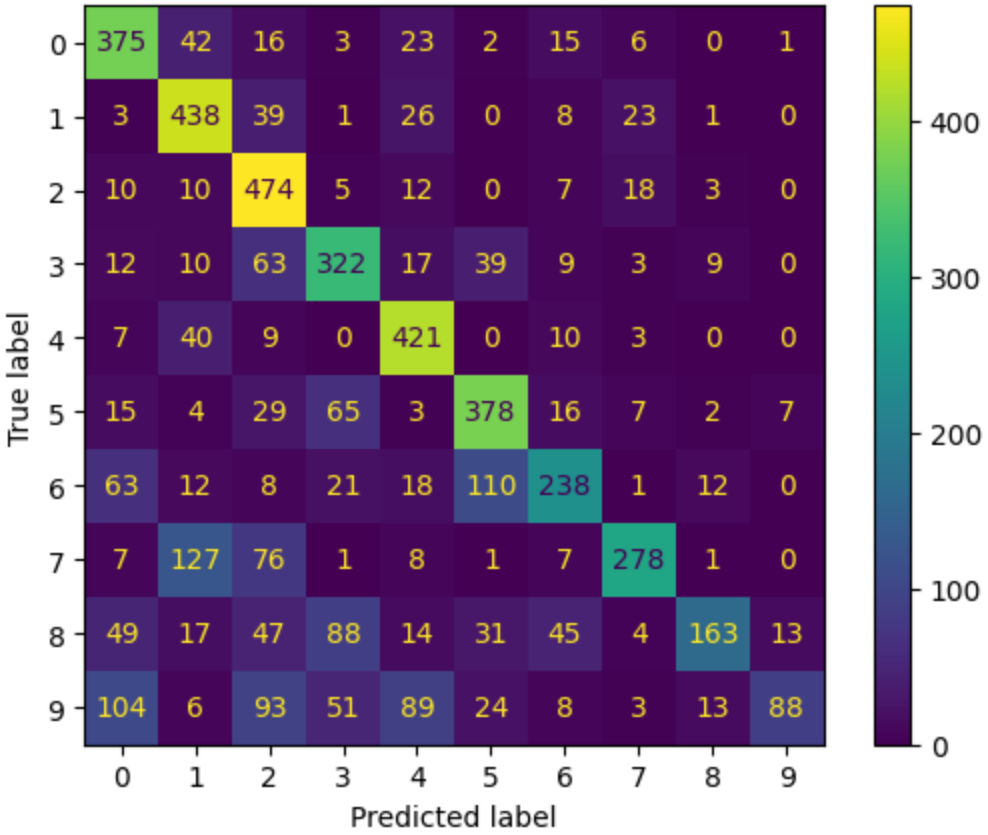
In [56]:
```
clf = RandomForestClassifier(max_depth=15, n_estimators=175, random_state=42)
clf.fit(features_train_3_copy[:2500], digits_train.iloc[features_train_3_copy.index
cm = confusion_matrix(digits_test_3, clf.predict(features_test_3), labels=clf.class
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
disp.plot()
plt.show()
```



In [58]:
```
features_3 = [features_train_3_copy.iloc[:2500], features_train_1_copy[:400], \
                        features_train_2_copy[:215], features_train_0_copy[:300], f
new_features_domain_3 = pd.concat(features_3)
to_append = [digits_train_3[:2500][0], digits_train_1[:400][0], digits_train_2[:215
new_digits_train_3 = pd.concat(to_append)
clf.fit(new_features_domain_3, new_digits_train_3)
new_predict_y_3 = clf.predict(features_test_3)
accuracy_score(digits_test_3, new_predict_y_3)
```

Out[58]: 0.635

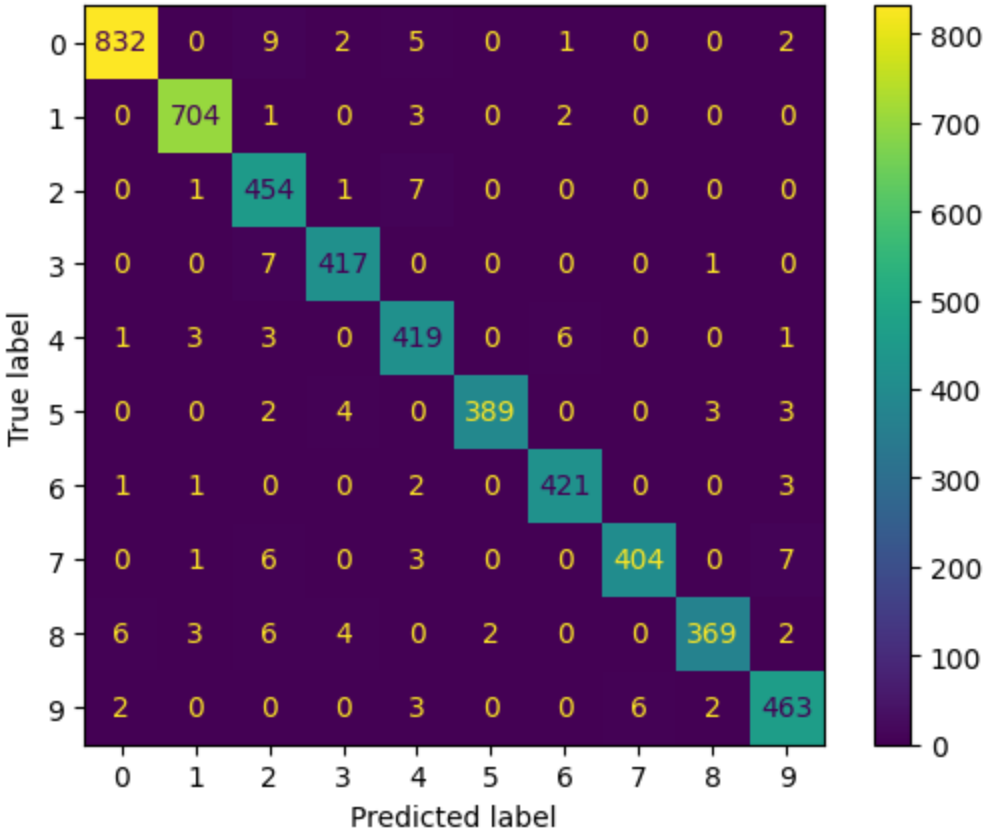
In [59]:
```
cm = confusion_matrix(digits_test_3, new_predict_y_3, labels=clf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
disp.plot()
plt.show()
```



Domain 4

previous accuracy was 97.44%

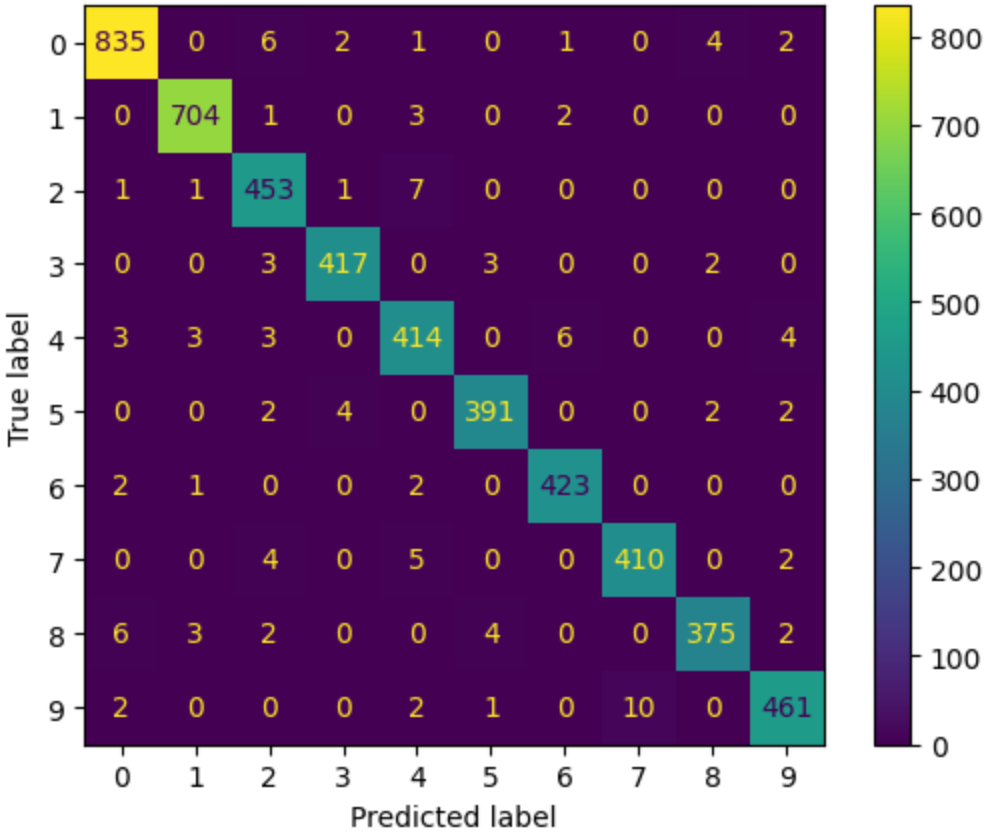
In [60]:
```
clf = RandomForestClassifier(max_depth=25, n_estimators=175, random_state=42)
clf.fit(features_train_4_copy[:2500], digits_train.iloc[features_train_4_copy.index
cm = confusion_matrix(digits_test_4, clf.predict(features_test_4), labels=clf.class
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
disp.plot()
plt.show()
```



In [63]:
```
features_4 = [features_train_4_copy.iloc[:2500], features_train_1_copy[:300], \
                        features_train_2_copy[:150], features_train_3_copy[:150], f
new_features_domain_4 = pd.concat(features_4)
to_append = [digits_train_4[:2500][0], digits_train_1[:300][0], digits_train_2[:150
new_digits_train_4 = pd.concat(to_append)
clf.fit(new_features_domain_4, new_digits_train_4)
new_predict_y_4 = clf.predict(features_test_4)
accuracy_score(digits_test_4, new_predict_y_4)
```

Out[63]: 0.9766

In [64]:
```
cm = confusion_matrix(digits_test_4, new_predict_y_4, labels=clf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
disp.plot()
plt.show()
```



Conclude

At part 1, we got 77.44% accuracy with whole data and domain(same weight)

At part 2, we got this accarcy scores from every single domain: (in perentage)

- domain 0 = 96.44
- domain 1 = 84.26
- domain 2 = 72.28
- domain 3 = 61.6
- domain 4 = 97.44

And Finally in the last part we improve scores by merging data from other domains. New Results are:

- domain 0 = 96.96 (+0.52%)
- domain 1 = 86.46 (+2.2%)
- domain 2 = 72.7 (+0.42%)
- domain 3 = 63.5 (+1.9%)
- domain 4 = 97.66 (+0.22%)