MNIST

MNIST Data Prediction Using Daimensions

MNIST is a well-known dataset of handwritten digits and a standard for machine learning models. Here, we test how Daimensions does on it.

0. Setup

We'll get the csv from the OpenML link and use a pandas dataframe to split it into training and validation data in csv's.

```
In [1]: # using pandas to get csv as a dataframe and see how it looks
        import pandas as pd
        from sklearn.model_selection import train_test_split

        dataset_url = 'https://www.openml.org/data/get_csv/52667/mnist_784.csv'
        data = pd.read_csv(dataset_url)
        data.describe()
```

```
        pixel1   pixel2   pixel3   pixel4   pixel5   pixel6   pixel7   pixel8
count  70000.0  70000.0  70000.0  70000.0  70000.0  70000.0  70000.0  70000.0
mean       0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
std        0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
min        0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
25%        0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
50%        0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
75%        0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
max        0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0

        pixel9  pixel10    ...        pixel783     pixel784        class
count  70000.0  70000.0    ...         70000.0      70000.0      70000.0
mean       0.0      0.0    ...        0.000000     0.000000     4.452429
std        0.0      0.0    ...        0.000000     0.000000     2.890195
min        0.0      0.0    ...        0.000000     0.000000     0.000000
25%        0.0      0.0    ...        0.000000     0.000000     2.000000
50%        0.0      0.0    ...        0.000000     0.000000     4.000000
75%        0.0      0.0    ...        0.000000     0.000000     7.000000
max        0.0      0.0    ...      254.000000   253.000000     9.000000

[8 rows x 785 columns]
```

```
In [2]: # split data into training and testing csv's, y is for the target column (int0)
        y = data['class']
        X = data.drop('class', axis=1)
        X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2)
        pd.concat([X_train, y_train], axis=1).to_csv('mnist_train.csv',index=False)
        pd.concat([X_test, y_test], axis=1).to_csv('mnist_valid.csv',index=False)
```

1. Get Measurements

We always want to measure our data before building our predictor in order to ensure we are building the right model. For more information about how to use Daimensions and why we want to measure our data beforehand, check out the Titanic notebook.

```
In [3]: ! btc -measureonly mnist_train.csv
```

```
Brainome Daimensions(tm) 0.97 Copyright (c) 2019, 2020 by Brainome, Inc. All Rights
     Reserved.

Data:
Number of instances: 56000
Number of attributes: 785
Number of classes: 10
Class balance: 9.91% 11.27% 9.97% 10.2% 9.7% 9.02% 9.81% 10.44% 9.71% 9.97%

Learnability:
Best guess accuracy: 11.27%
Capacity progression (# of decision points): [14, 15, 16, 16, 17, 17]
Decision Tree: 49973 parameters
Estimated Memory Equivalent Capacity for Neural Networks: 12746 parameters

Risk that model needs to overfit for 100% accuracy...
using Decision Tree: 99.15%
using Neural Networks: 100.00%

Expected Generalization...
using Decision Tree: 1.12 bits/bit
using a Neural Network: 4.39 bits/bit

Recommendations:
Note: Maybe enough data to generalize. [yellow]
Warning: Data has high information density. Expect varying results and increase --effort.`
```

2. Build the Predictor

Based on our measurements, Daimensions recommends we use a neural network (higher expected generalization) and more effort for this dataset.

```
In [5]: ! btc -vvv -f NN mnist_train.csv -o mnist_predict.py -e 5

Brainome Daimensions(tm) 0.97 Copyright (c) 2019, 2020 by Brainome, Inc. All Rights
    Reserved.

Input: mnist_train.csv

Architecting model...done.
Model capacity (MEC):   1075 bits
Architecture efficiency: 1.0 bits/parameter

Estimating time to prime model...done.
Estimated time to prime model: 0d 1h 10m 22s

Priming model...done.
Estimating training time...done.
Estimated training time: 0d 3h 36m 53s

Training...done.
Model created:
Sequential(
  (0): Linear(in_features=60, out_features=15, bias=True)
  (1): ReLU()
  (2): Linear(in_features=15, out_features=10, bias=True)
)

Compiling predictor...done.
Validating predictor...done.
Classifier Type:                Neural Network
System Type:                    10-way classifier
Best-guess accuracy:            11.23%
Model accuracy:                 68.28% (38239/56000 correct)
Improvement over best guess:    57.05% (of possible 88.77%)
Model capacity (MEC):           1075 bits
Generalization ratio:           35.57 bits/bit
Confusion Matrix:
 [8.04% 0.01% 0.07% 0.22% 0.05% 0.54% 0.77% 0.03% 0.05% 0.03%]
 [0.01% 10.37% 0.10% 0.11% 0.07% 0.41% 0.01% 0.03% 0.07% 0.04%]
 [0.43% 0.07% 6.75% 0.60% 0.39% 0.43% 0.78% 0.13% 0.35% 0.03%]
 [0.33% 0.40% 0.69% 5.88% 0.16% 1.17% 0.27% 0.12% 0.98% 0.23%]
 [0.10% 0.20% 0.20% 0.08% 6.45% 0.55% 0.45% 0.34% 0.32% 1.19%]
 [1.55% 0.47% 0.24% 0.88% 0.22% 3.90% 0.71% 0.05% 0.63% 0.43%]
 [1.22% 0.08% 0.61% 0.14% 0.14% 1.06% 6.40% 0.10% 0.07% 0.00%]
 [0.05% 0.13% 0.22% 0.18% 0.27% 0.21% 0.02% 8.28% 0.24% 0.81%]
 [0.35% 0.49% 0.40% 0.74% 0.08% 1.36% 0.21% 0.08% 5.36% 0.56%]
 [0.09% 0.40% 0.08% 0.17% 0.83% 0.40% 0.05% 0.74% 0.38% 6.85%]
```

```
Overfitting:                     No

Output: mnist_predict.py
READY.
```

3. Validate the Model

Now we can validate our model on a separate set of data that wasn't used for training.

```
In [6]: ! python3 mnist_predict.py -validate mnist_valid.csv
```

```
Classifier Type:                 Neural Network
System Type:                     10-way classifier
Best-guess accuracy:             11.33%
Model accuracy:                  68.15% (9542/14000 correct)
Improvement over best guess:     56.82% (of possible 88.67%)
Model capacity (MEC):            1075 bits
Generalization ratio:            8.87 bits/bit
Confusion Matrix:
 [8.26% 0.01% 0.06% 0.21% 0.06% 0.58% 0.82% 0.04% 0.05% 0.06%]
 [0.01% 10.56% 0.07% 0.09% 0.04% 0.42% 0.02% 0.03% 0.06% 0.02%]
 [0.48% 0.07% 6.72% 0.63% 0.54% 0.44% 0.76% 0.12% 0.28% 0.04%]
 [0.29% 0.36% 0.75% 5.97% 0.11% 1.15% 0.27% 0.11% 0.90% 0.21%]
 [0.08% 0.21% 0.21% 0.07% 6.07% 0.51% 0.41% 0.36% 0.34% 0.99%]
 [1.49% 0.45% 0.25% 0.81% 0.24% 3.71% 0.74% 0.04% 0.61% 0.46%]
 [1.29% 0.08% 0.54% 0.11% 0.16% 1.06% 6.45% 0.08% 0.04% 0.01%]
 [0.05% 0.15% 0.19% 0.13% 0.31% 0.20% 0.03% 8.33% 0.23% 0.86%]
 [0.41% 0.57% 0.37% 0.95% 0.10% 1.35% 0.17% 0.14% 5.51% 0.65%]
 [0.14% 0.38% 0.05% 0.20% 0.87% 0.27% 0.07% 0.84% 0.36% 6.56%]
```

Hooray! We finished building our model and validating its accuracy. We also have the confusion matrix, which compares the actual target classes (columns) with the predicted class (rows). Diagonal values are correctly predicted values.