USPS Data Prediction Using Daimensions

This dataset is from OpenML who describes the data as, "Normalized handwritten digits, automatically scanned from envelopes by the U.S. Postal Service."

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 [2]:

```
# 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/19329737/usps.csv'
data = pd.read_csv(dataset_url)
data.describe()
```

Out[2]:

	int0	double1	double2	double3	double4	double5	double6	double7	double8
count	9298.000000	9298.000000	9298.000000	9298.000000	9298.000000	9298.000000	9298.000000	9298.000000	9298.000000
mean	4.892020	-0.991800	-0.972226	-0.930421	-0.852805	-0.733673	-0.578239	-0.391187	-0.228260
std	3.001086	0.050814	0.118296	0.195285	0.284053	0.372653	0.435317	0.452878	0.454537
min	1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000
25%	2.000000	-1.000000	-1.000000	-1.000000	-0.999914	-0.996085	-0.963110	-0.787003	-0.620084
50%	5.000000	-1.000000	-0.999992	-0.999608	-0.991661	-0.932991	-0.747495	-0.447743	-0.138583
75%	7.000000	-0.999969	-0.998444	-0.979572	-0.861493	-0.589829	-0.260331	0.000547	0.143727
max	10.000000	0.000308	0.332928	0.479436	0.523534	0.527370	0.531509	0.531319	0.531368

8 rows × 257 columns

In [3]:

```
# split data into training and testing csv's, y is for the target column (int0)
y = data.int0
X = data.drop('int0', 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('usps_train.csv',index=False)
pd.concat([X_test, y_test], axis=1).to_csv('usps_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. Don't forget to use -target int0 because the target column is not on the very right for this dataset.

In [4]:

```
! ./btc -measureonly usps_train.csv -target int0
```

```
Brainome Daimensions(tm) 0.99 Copyright (c) 2019, 2020 by Brainome, Inc. All Rights Reser
ved.
```

```
Licensed to: Alexander Makhratchev
Expiration date: 2021-04-30 (64 days left)
Number of threads: 1
Maximum file size: 30720MB
Running locally.
WARNING: Could not detect a GPU. Neural Network generation will be slow.

Data:
Number of instances: 7438
Number of attributes: 256
Number of classes: 10
Class balance: 7.556% 8.86% 16.792% 13.7% 9.747% 8.672% 9.384% 9.075% 8.551% 7.663%

Learnability:
Best guess accuracy: 16.79%
Capacity progression: [11, 12, 13, 13, 14, 14]
Decision Tree: 5973 parameters
Estimated Memory Equivalent Capacity for Neural Networks: 3481 parameters

Risk that model needs to overfit for 100% accuracy...
using Decision Tree: 90.00%
using Neural Networks: 100.00%

Expected Generalization...
using Decision Tree: 4.07 bits/bit
using a Neural Network: 6.99 bits/bit

Recommendations:
Note: Maybe enough data to generalize. [yellow]
Warning: Remapped class labels to be contiguous. Use -cm if DET/ROC-based accuracy measur
ements are wrong.
Time estimate for a Neural Network:
Estimated time to architect: 0d 0h 1m 22s

Estimated time to prime (subject to change after model architecting): 0d 0h 49m 57s

Time estimate for Decision Tree:
Estimated time to prime a decision tree: a few seconds
```

2. Build the Predictor

Based on our measurements, Daimensions recommends we use a neural network (higher expected generalization) and more effort for this dataset. Don't forget to use -target because the target column isn't on the very right.

In [5]:

```
! ./btc -f NN usps_train.csv -o usps_predict.py -target int0 -e 5
```

```
Brainome Daimensions(tm) 0.99 Copyright (c) 2019, 2020 by Brainome, Inc. All Rights Reser
ved.
Licensed to: Alexander Makhratchev
Expiration date: 2021-04-30 (64 days left)
Number of threads: 1
Maximum file size: 30720MB
Running locally.
WARNING: Could not detect a GPU. Neural Network generation will be slow.

Data:
Number of instances: 7438
Number of attributes: 256
Number of classes: 10
Class balance: 7.556% 8.86% 16.792% 13.7% 9.747% 8.672% 9.384% 9.075% 8.551% 7.663%

Learnability:
Best guess accuracy: 16.79%
Capacity progression: [11, 12, 13, 13, 14, 14]
Decision Tree: 5973 parameters
Estimated Memory Equivalent Capacity for Neural Networks: 3481 parameters

Risk that model needs to overfit for 100%
```

```
Risk that model needs to overfit for 100% accuracy...
using Decision Tree: 90.00%
using Neural Networks: 100.00%

Expected Generalization...
using Decision Tree: 4.07 bits/bit
using a Neural Network: 6.99 bits/bit

Recommendations:
Note: Maybe enough data to generalize. [yellow]
Warning: Remapped class labels to be contiguous. Use -cm if DET/ROC-based accuracy measur
ements are wrong.
Time estimate for a Neural Network:
Estimated time to architect: 0d 0h 1m 13s

Estimated time to prime (subject to change after model architecting): 0d 0h 42m 59s

Note: Machine learner type NN given by user.
Model capacity (MEC):     286 bits
Architecture efficiency:   1.0 bits/parameter

Estimated time to prime model: 0d 0h 40m 40s


Estimated training time: 0d 2h 29m 57s


Classifier Type:                     Neural Network
System Type:                         10-way classifier
Training/Validation Split:           60:40%
Best-guess accuracy:                 16.79%
Training accuracy:                   99.43% (4437/4462 correct)
Validation accuracy:                 94.05% (2799/2976 correct)
Overall Model accuracy:              97.28% (7236/7438 correct)
Overall Improvement over best guess: 80.49% (of possible 83.21%)
Model capacity (MEC):                574 bits
Generalization ratio:                41.22 bits/bit
Model efficiency:                    0.14%/parameter
Confusion Matrix:
 [7.30% 0.05% 0.07% 0.00% 0.01% 0.04% 0.01% 0.00% 0.04% 0.03%]
 [0.05% 8.54% 0.01% 0.00% 0.01% 0.03% 0.00% 0.00% 0.01% 0.20%]
 [0.04% 0.01% 16.52% 0.00% 0.12% 0.00% 0.01% 0.07% 0.00% 0.01%]
 [0.01% 0.00% 0.00% 13.63% 0.01% 0.03% 0.01% 0.00% 0.00% 0.00%]
 [0.09% 0.00% 0.05% 0.00% 9.48% 0.00% 0.04% 0.03% 0.03% 0.03%]
 [0.04% 0.01% 0.01% 0.00% 0.00% 8.39% 0.09% 0.00% 0.09% 0.03%]
 [0.07% 0.00% 0.04% 0.04% 0.07% 0.07% 9.01% 0.04% 0.01% 0.04%]
 [0.04% 0.00% 0.01% 0.01% 0.01% 0.00% 0.05% 8.91% 0.00% 0.03%]
 [0.01% 0.01% 0.01% 0.01% 0.00% 0.13% 0.05% 0.00% 8.31% 0.00%]
 [0.04% 0.12% 0.13% 0.00% 0.03% 0.04% 0.04% 0.04% 0.03% 7.19%]
Generalization efficiency:           11.24
Overfitting:                         No
Note: Labels have been remapped to '9'=0, '4'=1, '1'=2, '2'=3, '3'=4, '10'=5, '5'=6, '7'=
7, '8'=8, '6'=9.
```

3. Validate the Model

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

In [6]:

```
! python3 usps_predict.py -validate usps_valid.csv
```

```
Classifier Type:                    Neural Network
System Type:                        10-way classifier
Best-guess accuracy:                16.34%
Model accuracy:                     93.49% (1739/1860 correct)
Improvement over best guess:        77.15% (of possible 83.66%)
Model capacity (MEC):               574 bits
Generalization ratio:               9.89 bits/bit
Model efficiency:                   0.13%/parameter
```

```
Confusion Matrix:
 [6.88% 0.22% 0.05% 0.00% 0.11% 0.05% 0.32% 0.05% 0.05% 0.11%]
 [0.05% 8.39% 0.00% 0.00% 0.05% 0.00% 0.00% 0.00% 0.05% 0.32%]
 [0.05% 0.05% 15.81% 0.00% 0.22% 0.00% 0.00% 0.05% 0.05% 0.11%]
 [0.05% 0.05% 0.00% 13.17% 0.05% 0.05% 0.00% 0.05% 0.00% 0.00%]
 [0.22% 0.05% 0.16% 0.00% 10.05% 0.00% 0.27% 0.05% 0.00% 0.16%]
 [0.22% 0.00% 0.00% 0.00% 0.11% 8.60% 0.27% 0.00% 0.22% 0.05%]
 [0.00% 0.00% 0.16% 0.00% 0.00% 0.27% 7.74% 0.00% 0.00% 0.11%]
 [0.05% 0.00% 0.05% 0.00% 0.11% 0.00% 0.16% 8.17% 0.00% 0.00%]
 [0.00% 0.11% 0.11% 0.00% 0.05% 0.11% 0.05% 0.00% 7.96% 0.00%]
 [0.11% 0.38% 0.22% 0.05% 0.00% 0.11% 0.11% 0.11% 0.05% 6.72%]
```

Hooray! We have validated the accuracy of our model and found that it has a 92.9% accuracy for the test data. We can also see the confusion matrix, which tells us the percentage of data points from each class (columns) that were predicted to be in a certain class (rows). The diagonals are correctly predicted data points.