USPS

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 [9]: # 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()
```

	int0	double1	double2	double3	double4
count	9298.000000	9298.000000	9298.000000	9298.000000	9298.000000
mean	4.892020	-0.991800	-0.972226	-0.930421	-0.852805
std	3.001086	0.050814	0.118296	0.195285	0.284053
min	1.000000	-1.000000	-1.000000	-1.000000	-1.000000
25%	2.000000	-1.000000	-1.000000	-1.000000	-0.999914
50%	5.000000	-1.000000	-0.999992	-0.999608	-0.991661
75%	7.000000	-0.999969	-0.998444	-0.979572	-0.861493
max	10.000000	0.000308	0.332928	0.479436	0.523534

	double5	...	double254	double255	double256
count	9298.000000	...	9298.000000	9298.000000	9298.000000
mean	-0.733673	...	-0.936784	-0.970873	-0.989597
std	0.372653	...	0.183444	0.120247	0.058028
min	-1.000000	...	-1.000000	-1.000000	-1.000000
25%	-0.996085	...	-1.000000	-1.000000	-1.000000
50%	-0.932991	...	-0.999771	-0.999996	-1.000000
75%	-0.589829	...	-0.979766	-0.998040	-0.999942
max	0.527370	...	0.470479	0.314115	-0.162598

[8 rows x 257 columns]

```
In [10]: # 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 [9]: ! btc -measureonly usps_train.csv -target int0

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

Data:
Number of instances: 7438
Number of attributes: 257
Number of classes: 10
Class balance: 13.87% 10.1% 16.68% 8.79% 8.82% 7.7% 7.72% 8.87% 8.44% 9.01%

Learnability:
Best guess accuracy: 16.68%
Capacity progression (# of decision points): [11, 12, 13, 13, 14, 14]
Decision Tree: 6491 parameters
Estimated Memory Equivalent Capacity for Neural Networks: 3494 parameters

Risk that model needs to overfit for 100% accuracy...
using Decision Tree: 96.98%
using Neural Networks: 100.00%

Expected Generalization...
using Decision Tree: 1.15 bits/bit
using a Neural Network: 2.13 bits/bit

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

Time estimate for a Neural Network:
Estimated time to architect: 0d 0h 0m 21s
Estimated time to prime (subject to change after model architecting): 0d 0h 43m 54s

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 [4]: ! btc -f NN usps_train.csv -o usps_predict.py -target int0 -e 5

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

Time estimate for a Neural Network:
Estimated time to architect: 0d 0h 0m 20s
Estimated time to prime (subject to change after model architecting): 0d 0h 44m 54s
Note: Machine learner type NN given by user.
Model capacity (MEC):  280 bits
Architecture efficiency: 1.0 bits/parameter

Estimated time to prime model: 0d 0h 38m 16s

Priming model...done.
Estimated training time: 0d 1h 53m 9s

Training...done.
Classifier Type:                Neural Network
System Type:                    10-way classifier
Best-guess accuracy:            16.81%
Model accuracy:                 95.65% (7115/7438 correct)
Improvement over best guess:    78.84% (of possible 83.19%)
Model capacity (MEC):           340 bits
Generalization ratio:           20.92 bits/bit
Confusion Matrix:
 [8.54% 0.07% 0.05% 0.01% 0.01% 0.07% 0.00% 0.00% 0.09% 0.03%]
 [0.05% 9.64% 0.08% 0.04% 0.08% 0.12% 0.00% 0.00% 0.07% 0.03%]
 [0.11% 0.04% 16.39% 0.03% 0.11% 0.01% 0.01% 0.00% 0.03% 0.08%]
 [0.00% 0.00% 0.00% 13.55% 0.00% 0.04% 0.03% 0.00% 0.01% 0.01%]
 [0.00% 0.07% 0.05% 0.04% 8.48% 0.12% 0.01% 0.04% 0.03% 0.16%]
 [0.00% 0.09% 0.05% 0.01% 0.23% 7.14% 0.04% 0.01% 0.05% 0.08%]
 [0.00% 0.03% 0.01% 0.00% 0.01% 0.04% 7.82% 0.24% 0.08% 0.01%]
 [0.00% 0.00% 0.00% 0.01% 0.01% 0.00% 0.26% 8.48% 0.17% 0.01%]
 [0.09% 0.01% 0.03% 0.04% 0.00% 0.00% 0.00% 0.17% 8.62% 0.00%]
 [0.04% 0.05% 0.11% 0.01% 0.24% 0.09% 0.00% 0.08% 0.04% 6.99%]
Overfitting:                    No
```

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 [5]: ! python3 usps_predict.py -validate usps_valid.csv
```

```
Classifier Type:                Neural Network
System Type:                    10-way classifier
Best-guess accuracy:            16.32%
Model accuracy:                 92.90% (1728/1860 correct)
Improvement over best guess:    76.58% (of possible 83.68%)
Model capacity (MEC):           340 bits
Generalization ratio:           5.08 bits/bit
Confusion Matrix:
 [8.28% 0.38% 0.32% 0.11% 0.05% 0.05% 0.00% 0.00% 0.05% 0.11%]
 [0.05% 8.76% 0.16% 0.00% 0.00% 0.27% 0.05% 0.00% 0.05% 0.16%]
 [0.05% 0.16% 15.75% 0.00% 0.00% 0.11% 0.00% 0.00% 0.11% 0.11%]
 [0.05% 0.00% 0.00% 13.49% 0.00% 0.11% 0.00% 0.00% 0.00% 0.00%]
 [0.00% 0.11% 0.00% 0.05% 7.47% 0.22% 0.00% 0.05% 0.05% 0.32%]
 [0.00% 0.11% 0.11% 0.00% 0.11% 6.56% 0.00% 0.05% 0.11% 0.16%]
 [0.00% 0.00% 0.00% 0.00% 0.05% 0.11% 8.66% 0.43% 0.32% 0.00%]
 [0.00% 0.00% 0.00% 0.05% 0.00% 0.11% 0.11% 7.90% 0.16% 0.00%]
 [0.05% 0.16% 0.16% 0.00% 0.00% 0.00% 0.11% 0.16% 9.25% 0.05%]
 [0.16% 0.27% 0.11% 0.00% 0.32% 0.05% 0.00% 0.11% 0.05% 6.77%]
```

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.