DATA CONDITIONING

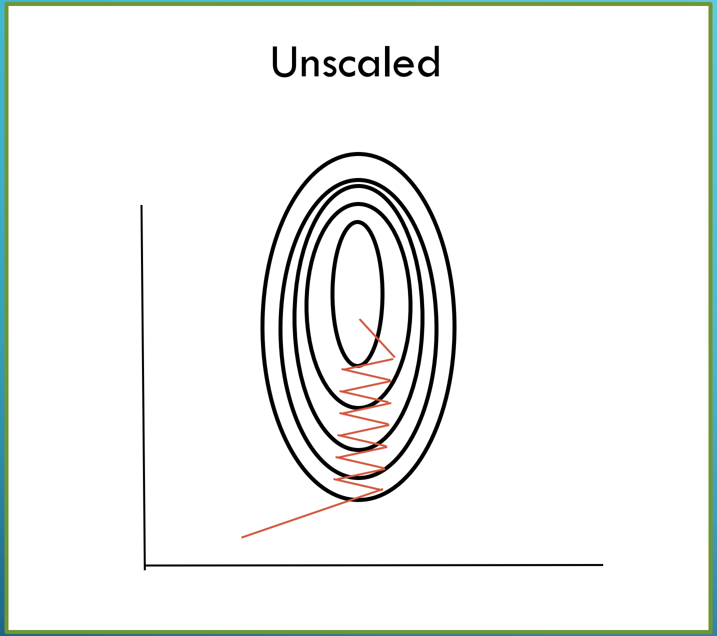
(MD-MLG) METROWEST DEVELOPERS – MACHINE LEARNING GROUP

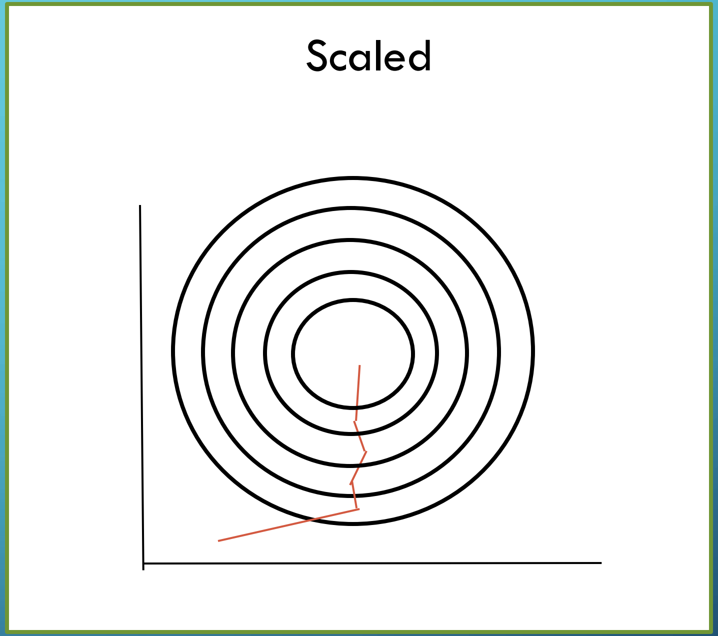
Gene Olafsen

FEATURE SCALING

- Scaling data provides a more efficient way for the ML algorithms to achieve an optimal solution faster.
- Gradient descent algorithms may spend more time 'bouncing' around trying to find a solution with unscaled data- then when the data is scaled.

SCALING





If the feature data is scaled, the contour of the cost function might be minimized- thus the gradient can assume a straighter path and achieve an optimal point faster. (Hopefully a global minimum)

FEATURE SCALING COMMON OPTIONS

- Normalization (min-max scaling)
  - Values are shifted and rescaled with a range of 0-1
- Standardization
  - Values are centered around a mean of 0
    - Much less affected by outliers.

SCIKIT-LEARN SCALING

- http://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html

ONE HOT ENCODE

- What: Used to transform categorical features into a format that is *digestible* by machine learning algorithms.
- List of Trades: Plumber, Electrician, Carpenter, Mason
- Many times, especially in code, developers will assign each an ordinal value - to be used in an enumeration.
  - 1 Plumber
  - 2 Electrician
  - 3 Carpenter
  - 4 Mason
- The ordinal number associated with each category holds no meaning in a machine learning context.

ONE HOT TRADE

- Create a feature for each category vote

Plumber	Electrician	Carpenter	Mason
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

ONE HOT ALTERNATIVE

- There are alternatives to One Hot encoding that doesn't blow out dimensionality of features.
- Binary coding can be used with acceptable results.

Trade	Feature 1	Feature 2	Feature 3
Plumber	0	0	1
Electrician	0	1	0
Carpenter	0	1	1
Mason	1	0	0
HVAC	1	0	1
Landscaper	1	1	0
Roofer	1	1	1

MISSING DATA

- Drop Row
- Median Value Replacement
- Drop Column

DROP ROW (ANY NAN)

Row	Feature A	Feature B
1	6.5	3.2
2	NaN	3.0
3	7.9	1.2

```
Pandas.DataFrame.dropna

df.dropna(axis=0, how='any')
```

Row	Feature A	Feature B
1	6.5	3.2
2	7.9	1.2

DROP ROW (ALL NAN)

Row	Feature A	Feature B
1	6.5	NaN
2	NaN	NaN
3	7.9	1.2

```
Pandas.DataFrame.dropna
```

```
df.dropna(axis=0, how='all')
```

Row	Feature A	Feature B
1	6.5	3.2
2	7.9	1.2

DROP COLUMN (ANY NAN)

Row	Feature A	Feature B
1	6.5	4.2
2	NaN	5.5
3	7.9	1.2

`Pandas.DataFrame.dropna`

```
df.dropna(axis=1, how='any')
```

Row	Feature B
1	4.2
2	5.5
3	1.2

DROP COLUMN (ANY NAN)

Row	Feature A	Feature B
1	NaN	4.2
2	NaN	5.5
3	NaN	Nan

```
Pandas.DataFrame.dropna

df.dropna(axis=1, how='all')
```

Row	Feature B
1	4.2
2	5.5
3	NaN

Feature B column remains because NOT 'all' entries are undefined.

PANDAS.DATAFRAME

- Two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns).
- Supported Operations:
    - Indexing, Iteration
    - Reindexing, Selection, Label Manipulation
    - Reshaping, Sorting, Transposing
    - Combining, Joining, Merging
    - Plotting
    - Serialization, IO, Conversion

FILLING NAN DATA

Row	Feature A	Feature B
1	NaN	4.2
2	NaN	5.5
3	NaN	Nan

```
Pandas.DataFrame.fillna

df.fillna(0, inplace=True)
```

Row	Feature A	Feature B
1	0	4.2
2	0	5.5
3	0	0

*The median value can be calculated for the column and used to fill NaN.

DROP COLUMNS

Row	Feature A	Feature B
1	2.3	4.2
2	8.1	5.5
3	NaN	3.0

```
Pandas.DataFrame.drop
```

```
df.drop("Feature A", axis=1)
```

Row	Feature B
1	4.2
2	5.5
3	0

TENSORFLOW AND NAN VALUES

- Different parts of TensorFlow treat them differently.
  - Float computations typically propagate them.
  - Some Int conversions (internally) treat them as 0.
  - Python parts of TensorFlow with Int computations often raise an "Nan" exception.

NUMPY

- A scientific computing package for Python providing a multidimensional array object and supporting routines.
  - mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation

NUMPY ARRAY CREATION

- At the core of the NumPy package, is the ndarray object. This encapsulates n-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance.

```
>>> import numpy
>>> a = numpy.zeros(shape=(2,2))
>>> a
Array([[0., 0.],
       [0., 0.]])

>>> a[0] = [2.4, 3.7]
>>> a
Array([[2.4, 3.7],
       [0., 0.]])
```

NUMPY ARRAY ATTRIBUTES

- The link below provides an interactive python prompt associated with examples.
- https://www.datacamp.com/community/tutorials/python-numpy-tutorial

```
In [1]: print(my_array)
[[1 2 3 4]
[5 6 7 8]]Click to add text

In [2]: print(my_array.strides)
(32, 8)

In [3]: print(my_array.dtype)
int64

In [4]: print(my_array.shape)
(2, 4)

In [5]: print(my_array.data)
<memory at 0x7f214d1b82d0>
```

NUMPY ARRAY ATTRIBUTES

```
# Print the number of `my_array`'s dimensions
print(my_array.ndim)

# Print the number of `my_array`'s elements
print(my_array.size)

# Print information about `my_array`'s memory layout
print(my_array.flags)

# Print the length of one array element in bytes
print(my_array.itemsize)

# Print the total consumed bytes by `my_array`'s elements
print(my_array.nbytes)
```

```
2
8
  C_CONTIGUOUS : True
  F_CONTIGUOUS : False
  OWNDATA : True
  WRITEABLE : True
  ALIGNED : True
  UPDATEIFCOPY : False
8
64
```

NUMPY BROADCASTING

- Broadcasting is the mechanism by which Numpy can perform numerical operations across arrays of different sizes.
- Here arrays of the SAME size are added.

```
# Initialize `x`
x = np.ones((3,4))

# Check shape of `x`
print(x.shape)

# Initialize `y`
y = np.random.random((3,4))

# Check shape of `y`
print(y.shape)

# Add `x` and `y`
x + y
```

```
(3, 4)
(3, 4)
Out[1]:
array([[ 1.04538982,  1.65032997,  1.24703754,  1.87061522],
       [ 1.23383259,  1.93691446,  1.76989044,  1.92724463],
       [ 1.62468318,  1.20556514,  1.94196648,  1.19122091]])
```

DIMENSION OF 1

- Broadcast operations also succeed if one dimension is 1.

```
# Initialize `x`
x = np.ones((3,4))

# Check shape of `x`
print(x.shape)

# Initialize `y`
y = np.arange(4)

# Check shape of `y`
print(y.shape)

# Subtract `x` and `y`
x - y
```

```
(3, 4)
(4,)
Out[1]:
array([[ 1.,  0., -1., -2.],
       [ 1.,  0., -1., -2.],
       [ 1.,  0., -1., -2.]])
```

NUMPY LOAD FROM FILE

- genfromtxt – method skips the first 'header' row and maps any NaN input to a value of –99.

```
# Sample text file data, where first row is a header:
# Value1 Value2 Value3
# 0.2839 0.3536 0.3661
# 0.1392 0.5875 NA
# 0.1581 0.2049 0.8628
# NA 0.5801 0.2038
# 0.5913 0.4367 0.7710

load_an_array = np.genfromtxt('data.txt', skip_header=1, filling_values=-99)
```

NUMPY SAVE TO FILE

- savetxt – creates a file named 'result.txt' with the contents of array 'x' inserting a delimiter of a comma between values.

```
np.savetxt(result.txt', x, delimiter=',')
```

CUSTOM TRANSFORMER

```
from sklearn.base import BaseEstimator, TransformerMixin

my_class_variable = false

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, pass_parameter = True): # no *args or **kargs
        self.new_pass_parameter = pass_parameter
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X, y=None):
        if self.new_pass_parameter:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household,
                             bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]

attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
    housing_extra_attribs = attr_adder.transform(housing.values)d text
```