Regression with Multiple Features

Multiple Linear Regression

- y = b0 + b1x1 + b2x2 + b3x3 + .. + bnxn
- We are still computing a linear equation, this time with multiple features
- Classic problem: predict housing prices using a variety of factors:
  - square footage
  - crime rate
  - nearness to food stores
  - etc.

sklearn LinearRegression

- Takes numpy arrays for features and target

```
from sklearn.linear_model import
  LinearRegression
```

- features should be in a numpy array of shape (samples, features) e.g. (506,13)
- target : numpy array of one row, shape=(13,0)

Boston Housing DataSet

```
from sklearn.datasets     import load_boston

boston = load_boston()
# set up features and target values
features = boston.data
target   = boston.target

```

```python
# what are the datatypes?
print (type(features))
print (type(target))

# how many features?
print (features.shape)
print (target.shape)
```

```
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
(506, 13)
(506,)
```

```python
# set up Linear Regression for multiple features
from sklearn.linear_model import LinearRegression

# create Linear Regression instance
bostonReg = LinearRegression()

# generate the model
model = bostonReg.fit(features, target)

# print results of model computation
print (model.coef_)
print (model.intercept_)

# what is variance accounted for by the model?
print ("Variance accounted for by model")
score_model13 = model.score(features, target)
print (score_model13)

```

```
[-1.07170557e-01  4.63952195e-02  2.08602395e-02  2.68856140e+00
 -1.77957587e+01  3.80475246e+00  7.51061703e-04 -1.47575880e+00
  3.05655038e-01 -1.23293463e-02 -9.53463555e-01  9.39251272e-03
 -5.25466633e-01]
36.49110328036133
Variance accounted for by model
0.7406077428649428
```

What is the change in variance when you remove one feature?

removing one (last) feature

```
# do regression for 12 features
model = bostonReg.fit(features12, target)

# what is variance accounted for by the model?
print ("Variance accounted for by model")
score_model12 = model.score(features12, target)
print (score_model12)

# what is percent difference?
diff = (score_model13 - score_model12) / score_model13
print ("difference  = ", diff)

```

```
Variance accounted for by model
0.6839521119105445
difference =  0.07649883693523579
```

Remove features that are highly correlated

Removing Highly Correlated Features

Use a correlation matrix to find highly correlated features

1. Convert feature matrix to DataFrame
2. Create correlation matrix: df.corr().abs()
3. Select upper triangle of correlation matrix
4. Find the index of the features with corr > 95%
5. Drop the features
6. Recompute – compare results

Create Upper triangular matrix

numpy.triu

numpy.triu(*m*, *k=0*)

Upper triangle of an array.

Return a copy of a matrix with the elements below the *k*-th diagonal zeroed.

```
# creating upper triangle matrix
m1 = np.array([[1,2,3,4,5], [6,7,8,9,10], [22,33,44,55,66], [11,23,34,45,56], [54,43,76,54,43]])
dfupper = np.triu(m1, k=1)
dfupper
```

```
array([[ 0,  2,  3,  4,  5],
       [ 0,  0,  8,  9, 10],
       [ 0,  0,  0, 55, 66],
       [ 0,  0,  0,  0, 56],
       [ 0,  0,  0,  0,  0]])
```

pandas.DataFrame.where

`DataFrame.where`(*cond*, *other=nan*, *inplace=False*, *axis=None*, *level=None*, *errors='raise'*, *try_cast=False*, *raise_on_error=None*) [source]

Return an object of same shape as self and whose corresponding entries are from self where *cond* is True and otherwise are from *other*.

The where method is an application of the if-then idiom. For each element in the calling DataFrame, if `cond` is `True` the element is used; otherwise the corresponding element from the DataFrame `other` is used.

The signature for `DataFrame.where()` differs from `numpy.where()`. Roughly `df1.where(m, df2)` is equivalent to `np.where(m, df1, df2)`.

Examples

```
>>> s = pd.Series(range(5))
>>> s.where(s > 0)
0    NaN
1    1.0
2    2.0
3    3.0
4    4.0
```

```
>>> s.where(s > 1, 10)
0    10.0
1    10.0
2     2.0
3     3.0
4     4.0
```

Setup DataFrame for correlation computation

```python
import pandas as pd
import numpy as np

# create dataframe from features
dfcorr = pd.DataFrame(features)
dfcorr.head(3)
```

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03

Create Correlation Matrix from Dataframe

```python
# create correlation matrix
matrix = dfcorr.corr().abs()
matrix.head(3)
```

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	1.000000	0.199458	0.404471	0.055295	0.417521	0.219940	0.350784	0.377904	0.622029	0.579564	0.288250	0.377365	0.452220
1	0.199458	1.000000	0.533828	0.042697	0.516604	0.311991	0.569537	0.664408	0.311948	0.314563	0.391679	0.175520	0.412995
2	0.404471	0.533828	1.000000	0.062938	0.763651	0.391676	0.644779	0.708027	0.595129	0.720760	0.383248	0.356977	0.603800

We want an upper triangular matrix. how?

```
# creating upper triangle matrix
m1 = np.array([[1,2,3,4,5], [6,7,8,9,10], [22,33,44,55,66], [11,23,34,45,56], [54,43,76,54,43]])
dfupper = np.triu(m1, k=1)
print (dfupper)


# explore .astype(np.bool)
m9 = dfupper.astype(np.bool)
m9

```

```
[[ 0  2  3  4  5]
 [ 0  0  8  9 10]
 [ 0  0  0 55 66]
 [ 0  0  0  0 56]
 [ 0  0  0  0  0]]
```

```
array([[False,  True,  True,  True,  True],
       [False, False,  True,  True,  True],
       [False, False, False,  True,  True],
       [False, False, False, False,  True],
       [False, False, False, False, False]])
```

pandas.DataFrame.where

`DataFrame.where`(*cond*, *other=nan*, *inplace=False*, *axis=None*, *level=None*, *errors='raise'*, *try_cast=False*, *raise_on_error=None*) [source]

Return an object of same shape as self and whose corresponding entries are from self where *cond* is True and otherwise are from *other*.

The where method is an application of the if-then idiom. For each element in the calling DataFrame, if `cond` is `True` the element is used; otherwise the corresponding element from the DataFrame `other` is used.

The signature for `DataFrame.where()` differs from `numpy.where()`. Roughly `df1.where(m, df2)` is equivalent to `np.where(m, df1, df2)`.

Examples

```
>>> s = pd.Series(range(5))
>>> s.where(s > 0)
0    NaN
1    1.0
2    2.0
3    3.0
4    4.0
```

```
>>> s.where(s > 1, 10)
0    10.0
1    10.0
2    2.0
3    3.0
4    4.0
```

```python
matrix_upper = matrix.where(np.triu(np.ones(matrix.shape), k=1).astype(np.bool))
matrix_upper
```

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	NaN	0.199458	0.404471	0.055295	0.417521	0.219940	0.350784	0.377904	0.622029	0.579564	0.288250	0.377365	0.452220
1	NaN	NaN	0.533828	0.042697	0.516604	0.311991	0.569537	0.664408	0.311948	0.314563	0.391679	0.175520	0.412995
2	NaN	NaN	NaN	0.062938	0.763651	0.391676	0.644779	0.708027	0.595129	0.720760	0.383248	0.356977	0.603800
3	NaN	NaN	NaN	NaN	0.091203	0.091251	0.086518	0.099176	0.007368	0.035587	0.121515	0.048788	0.053929
4	NaN	NaN	NaN	NaN	NaN	0.302188	0.731470	0.769230	0.611441	0.668023	0.188933	0.380051	0.590879
5	NaN	NaN	NaN	NaN	NaN	NaN	0.240265	0.205246	0.209847	0.292048	0.355501	0.128069	0.613808
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.747881	0.456022	0.506456	0.261515	0.273534	0.602339
7	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.494588	0.534432	0.232471	0.291512	0.496996
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.910228	0.464741	0.444413	0.488676
9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.460853	0.441808	0.543993
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.177383	0.374044
11	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.366087
12	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Which features do we eliminate?

Which features do we eliminate?

```
1 drop_values = [column for column in matrix_upper.columns if any(matrix_upper[column] > 0.90)]
2 drop_values
```

```
[9]
```

How to remove a column from a DataFrame?

How to remove a column from a DataFrame?

```
# drop feature from data frame
dfadjusted = dfcorr.drop(dfcorr.columns[drop_values], axis=1)
dfadjusted.head()
```

	0	1	2	3	4	5	6	7	8	10	11	12
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	18.7	396.90	5.33

Extract numpy array from dataframe

Extract numpy array from dataframe

```
# get features from new dataframe
features2 = dfadjusted.values
print (features2.shape)
```

```
(506, 12)
```

Compute New Regression using reduced feature list

```
# do regression for 12 features
reg2 = LinearRegression()

reduced_model = reg2.fit(features2, target)

# what is variance accounted for by the model?
print ("Variance accounted for by model")
score2 = reduced_model.score(features2, target)
print (score2)


```

```
Variance accounted for by model
0.7349412039707595
```