W4995 Applied Machine Learning

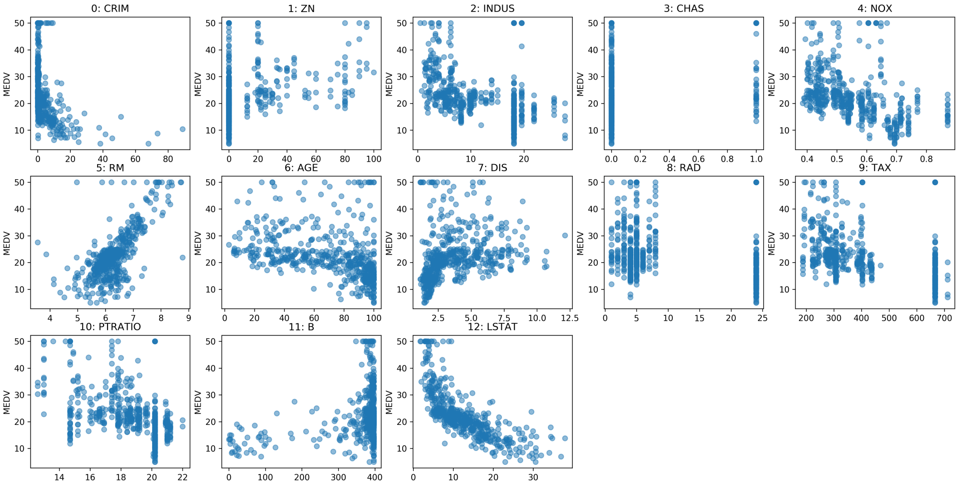
Preprocessing and Feature Transformations

02/06/19

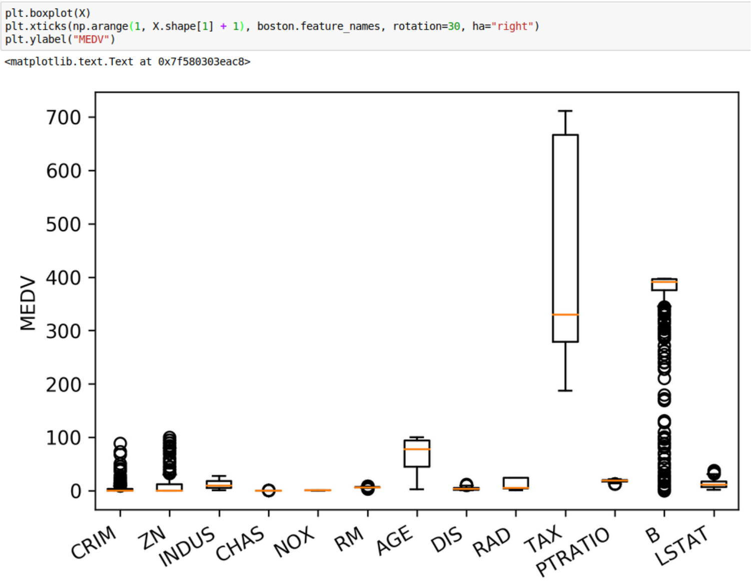
Andreas C. Müller

(Adapted and modified for CC 6021236 @ PCC/Ciencias/UCV by Eugenio Scalise, September 2019)

Boston Housing Dataset (scikit-learn)



Scaling

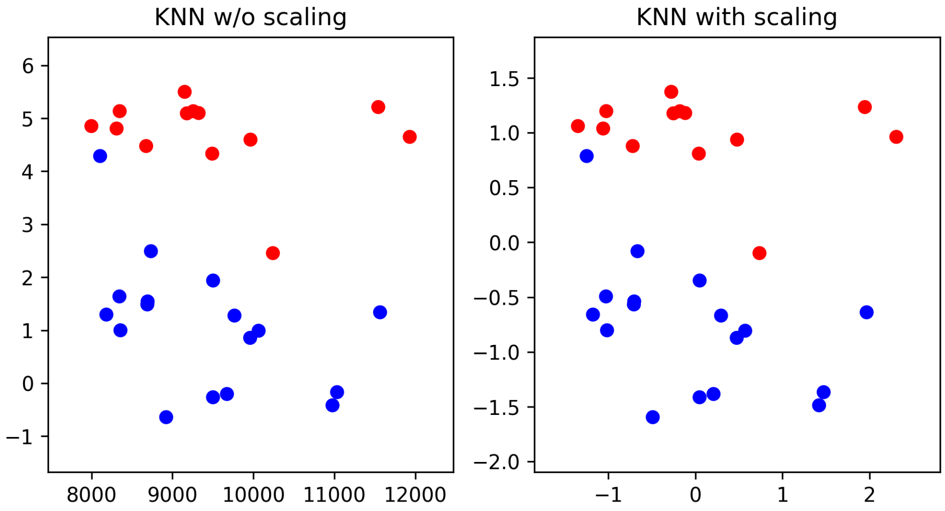


```
plt.boxplot(X)
plt.xticks(np.arange(1, X.shape[1] + 1), boston.feature_names, rotation=30, ha="right")
plt.ylabel("MEDV")
```

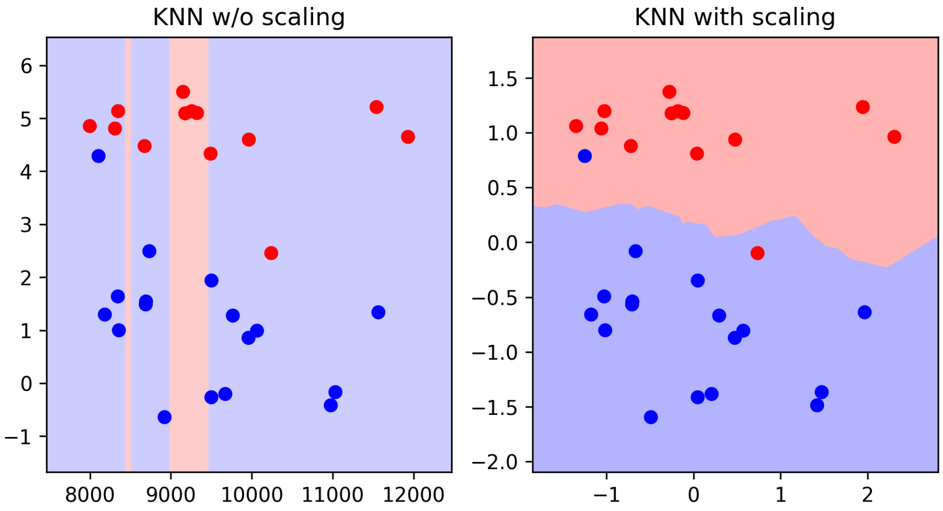
```
<matplotlib.text.Text at 0x7f580303eac8>
```

More about Boxplots: https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51

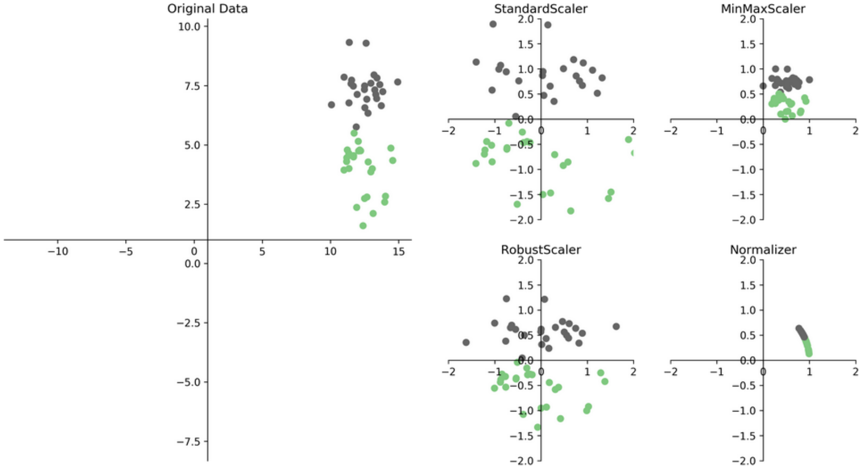
Scaling and Distances



Scaling and Distances



Ways to Scale Data



Scalers (scikit-learn)

- StandardScaler: ensures that for each feature the mean is 0 and the variance is 1, bringing all features to the same magnitude. However, this scaling does not ensure any particular minimum and maximum values for the features.
- RobustScaler: similar to the StandardScaler, however, it uses the median and quartiles instead of mean and variance. This makes the RobustScaler ignore data points that are very different from the rest (outliers).
- MinMaxScaler: shifts the data such that all features are exactly between 0 and 1.
- Normalizer : it sprojects a data point on the circle (or sphere) with a radius of 1. This is often used when only the direction (or angle) of the data matters, not the length of the feature vector.

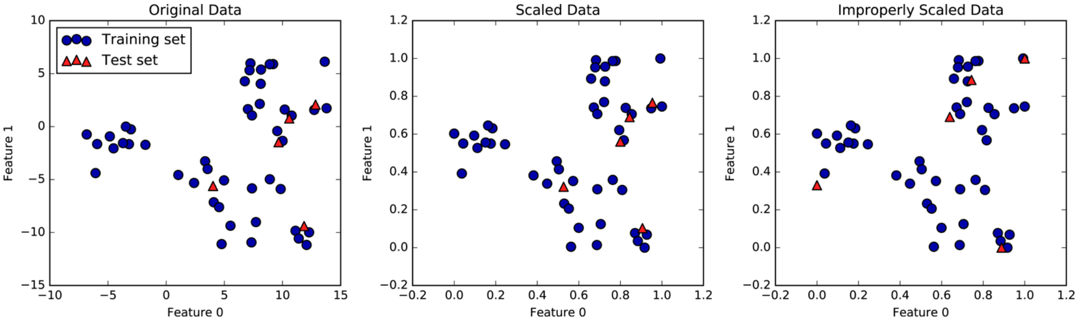
Standard Scaler Example

```
from sklearn.linear_model import Ridge
X, y = boston.data, boston.target
X_train, X_test, y_train, y_test = train_test_split(
    X, y, random_state=0)

scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)

ridge = Ridge().fit(X_train_scaled, y_train)
X_test_scaled = scaler.transform(X_test)
ridge.score(X_test_scaled, y_test)
```

```
0.634
```



Data Leakage: the scaling should happen inside the cross-validation loop, not outside.

Note: Read about pipelines in scikit-learn.

Discrete features

Categorical Variables

```
import pandas as pd
df = pd.DataFrame(
    {'boro': ['Manhattan', 'Queens', 'Manhattan', 'Brooklyn', 'Brooklyn', 'Bronx'],
     'vegan': ['No', 'No','No','Yes', 'Yes', 'No']})
```

	boro	vegan
0	Manhattan	No
1	Queens	No
2	Manhattan	No
3	Brooklyn	Yes
4	Brooklyn	Yes
5	Bronx	No

Ordinal encoding

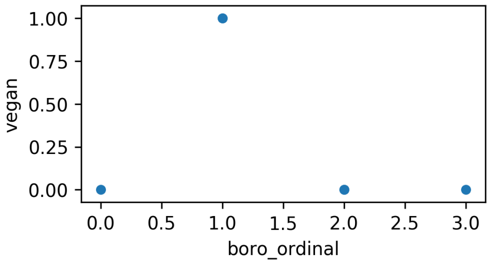
```
df['boro_ordinal'] = df.boro.astype("category").cat.codes
df
```

	boro	boro_ordinal	vegan
0	Manhattan	2	No
1	Queens	3	No
2	Manhattan	2	No
3	Brooklyn	1	Yes
4	Brooklyn	1	Yes
5	Bronx	0	No

Ordinal encoding

```
df['boro_ordinal'] = df.boro.astype("category").cat.codes
df
```

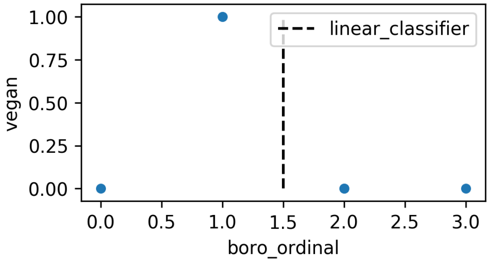
	boro	boro_ordinal	vegan
0	Manhattan	2	No
1	Queens	3	No
2	Manhattan	2	No
3	Brooklyn	1	Yes
4	Brooklyn	1	Yes
5	Bronx	0	No



Ordinal encoding

```
df['boro_ordinal'] = df.boro.astype("category").cat.codes
df
```

	boro	boro_ordinal	vegan
0	Manhattan	2	No
1	Queens	3	No
2	Manhattan	2	No
3	Brooklyn	1	Yes
4	Brooklyn	1	Yes
5	Bronx	0	No



- If you encode all three values using the same feature, then you are imposing a linear relation between them, and in particular you define an order between the categories.

One-Hot (Dummy) Encoding

	boro	vegan
0	Manhattan	No
1	Queens	No
2	Manhattan	No
3	Brooklyn	Yes
4	Brooklyn	Yes
5	Bronx	No

```
pd.get_dummies(df)
```

	boro_Bronx	boro_Brooklyn	boro_Manhattan	boro_Queens	vegan_No	vegan_Yes
0	0	0	1	0	1	0
1	0	0	0	1	1	0
2	0	0	1	0	1	0
3	0	1	0	0	0	1
4	0	1	0	0	0	1
5	1	0	0	0	1	0

One-Hot (Dummy) Encoding

	boro	vegan
0	Manhattan	No
1	Queens	No
2	Manhattan	No
3	Brooklyn	Yes
4	Brooklyn	Yes
5	Bronx	No

```
pd.get_dummies(df, columns=['boro'])
```

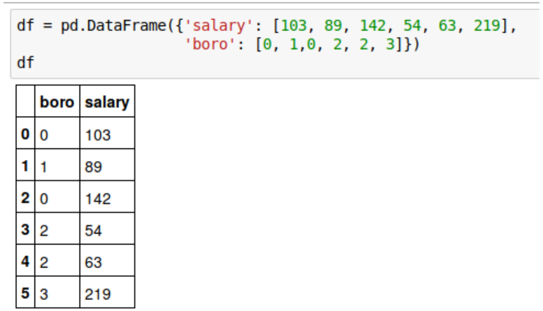
	vegan	boro_Bronx	boro_Brooklyn	boro_Manhattan	boro_Queens
0	No	0	0	1	0
1	No	0	0	0	1
2	No	0	0	1	0
3	Yes	0	1	0	0
4	Yes	0	1	0	0
5	No	1	0	0	0

One-Hot (Dummy) Encoding

	boro	vegan
0	2	No
1	3	No
2	2	No
3	1	Yes
4	1	Yes
5	0	No

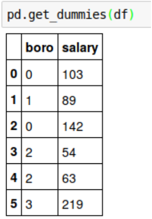
```
pd.get_dummies(df_ordinal, columns=['boro'])
```

	vegan	boro_0	boro_1	boro_2	boro_3
0	No	0	0	1	0
1	No	0	0	0	1
2	No	0	0	1	0
3	Yes	0	1	0	0
4	Yes	0	1	0	0
5	No	1	0	0	0



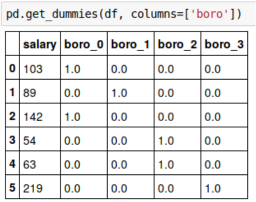
```
df = pd.DataFrame({'salary': [103, 89, 142, 54, 63, 219],
                   'boro': [0, 1,0, 2, 2, 3]})
df
```

	boro	salary
0	0	103
1	1	89
2	0	142
3	2	54
4	2	63
5	3	219



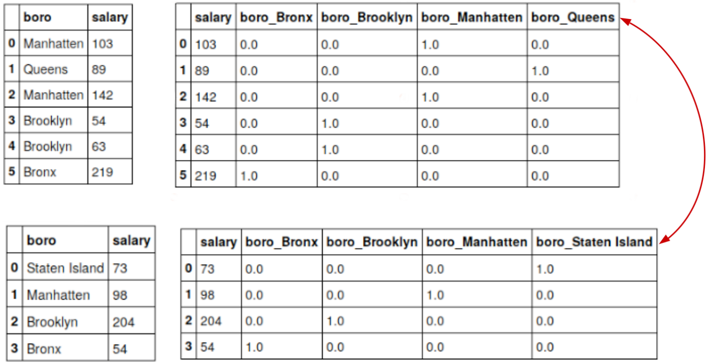
```
pd.get_dummies(df)
```

	boro	salary
0	0	103
1	1	89
2	0	142
3	2	54
4	2	63
5	3	219



```
pd.get_dummies(df, columns=['boro'])
```

	salary	boro_0	boro_1	boro_2	boro_3
0	103	1.0	0.0	0.0	0.0
1	89	0.0	1.0	0.0	0.0
2	142	1.0	0.0	0.0	0.0
3	54	0.0	0.0	1.0	0.0
4	63	0.0	0.0	1.0	0.0
5	219	0.0	0.0	0.0	1.0



	boro	salary
0	Manhatten	103
1	Queens	89
2	Manhatten	142
3	Brooklyn	54
4	Brooklyn	63
5	Bronx	219

	salary	boro_Bronx	boro_Brooklyn	boro_Manhatten	boro_Queens
0	103	0.0	0.0	1.0	0.0
1	89	0.0	0.0	0.0	1.0
2	142	0.0	0.0	1.0	0.0
3	54	0.0	1.0	0.0	0.0
4	63	0.0	1.0	0.0	0.0
5	219	1.0	0.0	0.0	0.0

	boro	salary
0	Staten Island	73
1	Manhatten	98
2	Brooklyn	204
3	Bronx	54

	salary	boro_Bronx	boro_Brooklyn	boro_Manhatten	boro_Staten Island
0	73	0.0	0.0	0.0	1.0
1	98	0.0	0.0	1.0	0.0
2	204	0.0	1.0	0.0	0.0
3	54	1.0	0.0	0.0	0.0

Pandas Categorical Columns

```python
import pandas as pd
df = pd.DataFrame({'salary': [103, 89, 142, 54, 63, 219],
                   'boro': ['Manhattan', 'Queens', 'Manhattan',
                            'Brooklyn', 'Brooklyn', 'Bronx']})

df['boro'] = pd.Categorical(df.boro, categories=['Manhattan', 'Queens', 'Brooklyn',
                                                'Bronx', 'Staten Island'])
pd.get_dummies(df)
```

	salary	boro_Manhattan	boro_Queens	boro_Brooklyn	boro_Bronx	boro_Staten Island
0	103	1	0	0	0	0
1	89	0	1	0	0	0
2	142	1	0	0	0	0
3	54	0	0	1	0	0
4	63	0	0	1	0	0
5	219	0	0	0	1	0

More encodings for categorical features:

http://contrib.scikit-learn.org/categorical-encoding/