Unsupervised Learning – Part 1

ESM3081 Programming for Data Science

Seokho Kang

성균관대학교
SUNG KYUN KWAN UNIVERSITY(SKKU)

Unsupervised Learning

Unsupervised Learning

- **Unsupervised Learning**
  - *(in general)* **Unlabeled training dataset** $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n\}$, where each data point $\boldsymbol{x}_i = (x_{i1}, \dots, x_{id})$ contains *d* feature values
  - To find useful properties/patterns of the structures of the dataset

Unsupervised Learning

- ***Unlabeled Dataset***

Column: variable, attribute, **feature**, …

Input

Label

Row:
data point, **instance**, example, record, pattern, object, …

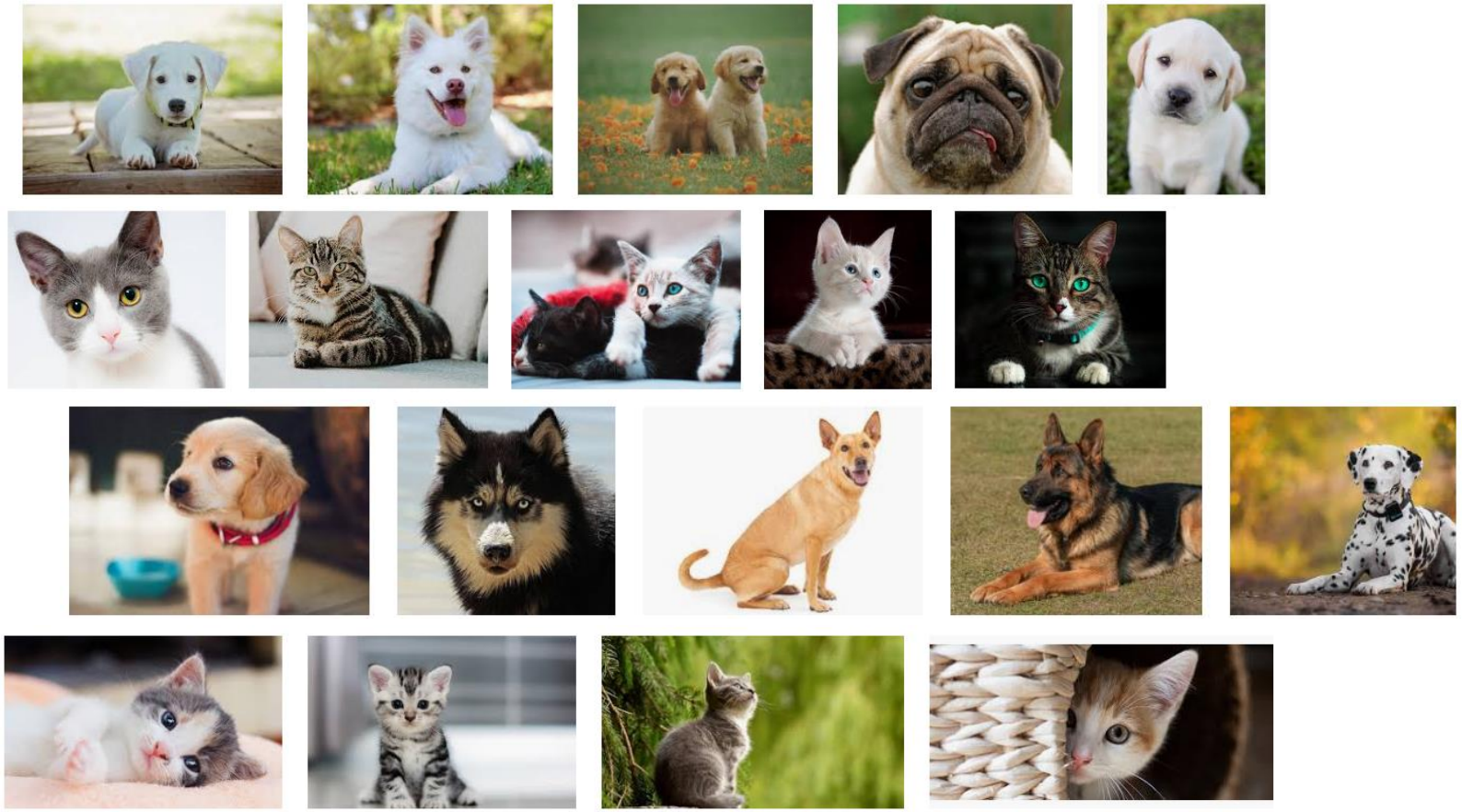
id	X_1	X_2	X_3	…	X_d
1	x_{11}	x_{12}	x_{13}	…	x_{1d}
2	x_{21}	x_{22}	x_{23}	…	x_{2d}
3	x_{31}	x_{32}	x_{33}	…	x_{3d}
4	x_{41}	x_{42}	x_{43}	…	x_{4d}
5	x_{51}	x_{52}	x_{53}	…	x_{5d}
6	x_{61}	x_{62}	x_{63}	…	x_{6d}
7	x_{71}	x_{72}	x_{73}	…	x_{7d}
…	…	…	…	…	…

X

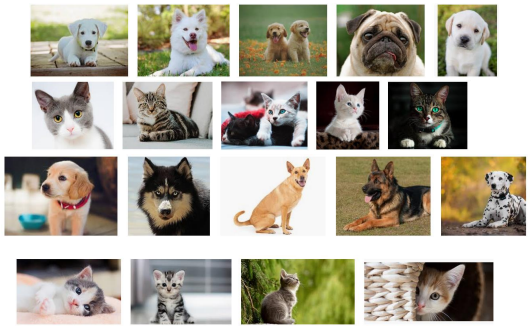
Unsupervised Learning

- Only the inputs are known, and no known outputs are given
- The unsupervised learning algorithm is just shown the input data and asked to extract knowledge from this data
- Unsupervised learning algorithms are usually harder to understand and evaluate

Types of Unsupervised Learning

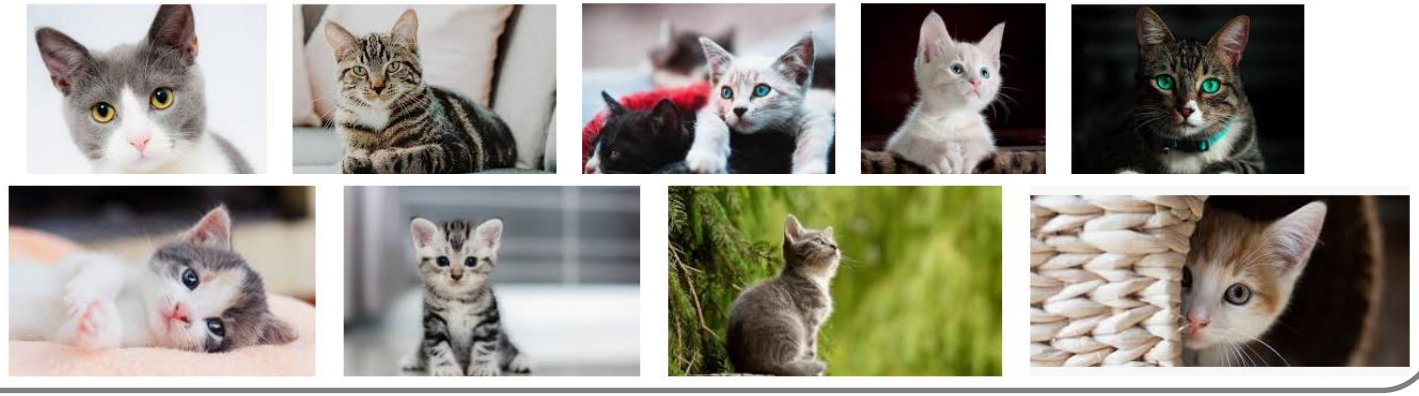


Types of Unsupervised Learning



Cluster 1

Cluster 2



Types of Unsupervised Learning

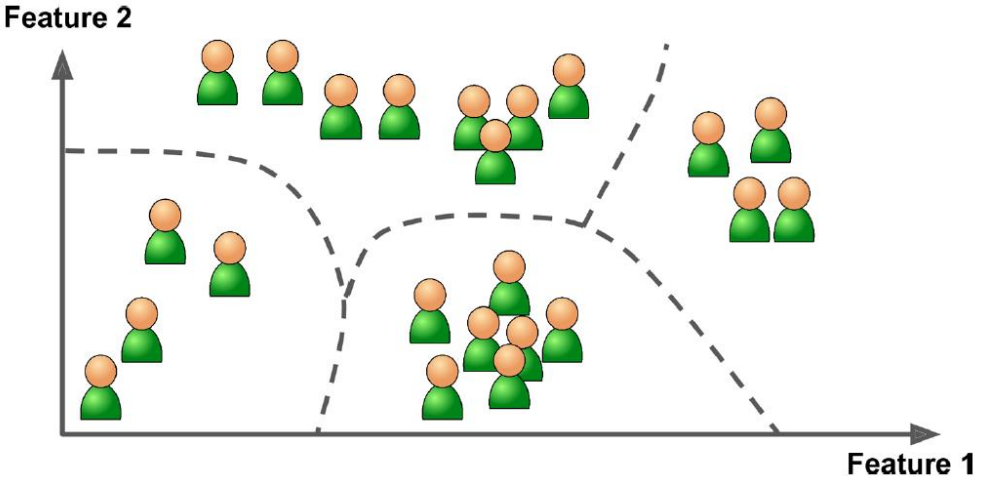
Types of Unsupervised Learning

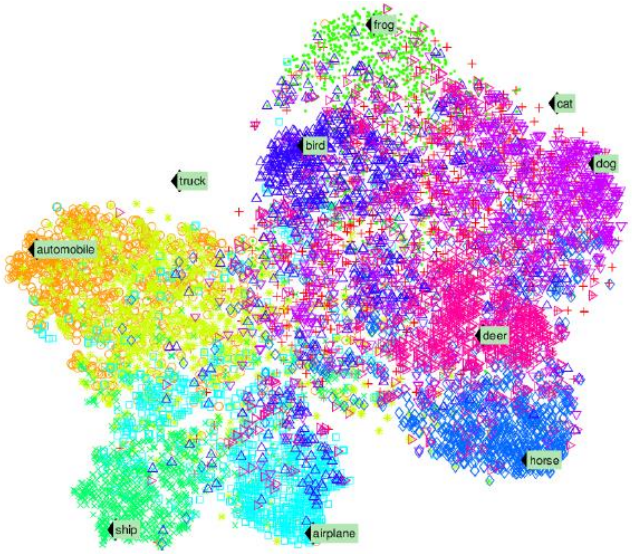
???

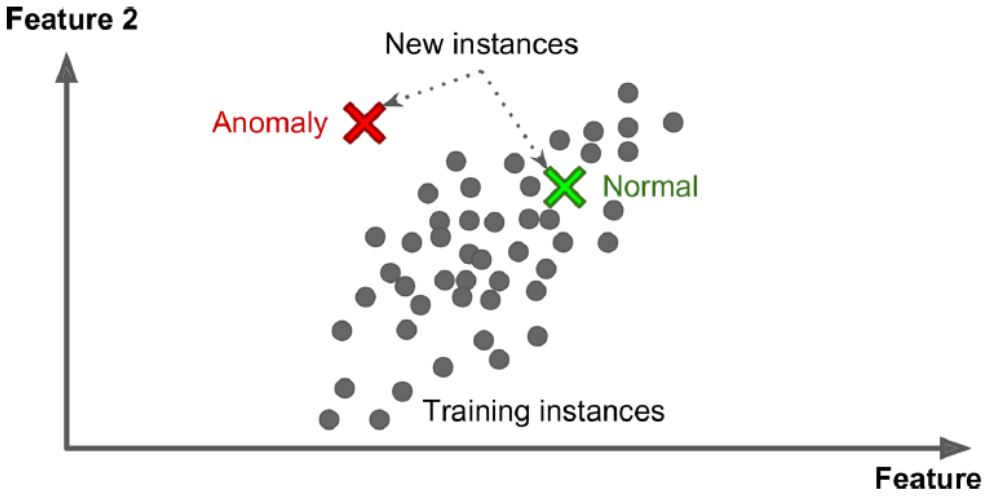
Types of Unsupervised Learning

- **Dimensionality Reduction**
  - Find a new way to represent this data that summarizes the essential characteristics with fewer features.
  - *Dimensionality reduction for visualization*: reduce to two or three dimensions for visualization purposes
- **Clustering**
  - Partition data into distinct groups of similar data points.
- **Anomaly Detection (One-Class Classification), Association Analysis, …**

Types of Unsupervised Learning







Challenges in Unsupervised Learning

- A major challenge in unsupervised learning is evaluating whether the algorithm learned something useful.
  - We don’t know what the right output should be.
  - It is very hard to tune the hyperparameters of an unsupervised learning algorithm.
  - The only way to evaluate the result is to inspect it manually.

- Unsupervised algorithms are used often in an exploratory setting
  - When a data scientist wants to understand the data better.
  - Rather than as part of a larger automatic system.

Learning algorithms covered in this course

- **Data Preprocessing and Scaling**

- **Unsupervised Learning**
  - **Dimensionality Reduction & Visualization**
    - (Projection) Principal Component Analysis (PCA)
    - (Manifold Learning) t-distributed Stochastic Neighbor Embedding (t-SNE)
    - …
  - **Clustering**
    - K-Means
    - Hierarchical Clustering
    - DBSCAN
    - …

Data Preprocessing and Scaling

Unsupervised Transformation of Data

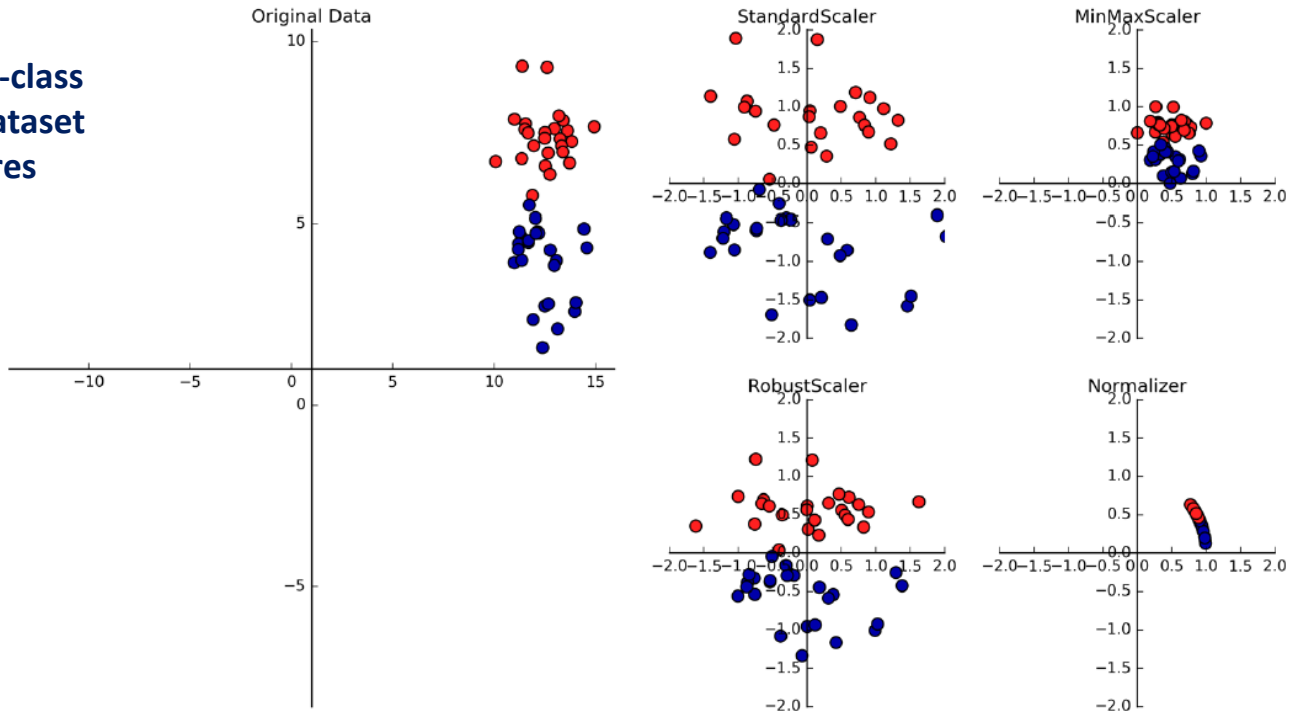
- **Unsupervised Transformation of Data**
  - Create a new representation of the data which might be easier for ***humans*** or ***machine learning algorithms*** to understand compared to the original representation of the data.
  - Find the parts or components that "make up" the data.

- A common application for unsupervised transformation of data is as a ***preprocessing step for supervised learning.***
  - Some supervised learning algorithms, like *k*-nearest neighbors, support vector machines, and neural networks are very sensitive to the scaling of the data.
  - Can sometimes improve the accuracy of supervised algorithms and lead to reduced memory and time consumption.

Different Kinds of Preprocessing

- A common practice is to adjust the features so that the data representation is more suitable for these algorithms.
- Often, this is a simple per-feature rescaling and shift of the data.
  - *StandardScaler* ensures that for each feature the mean is 0 and the variance is 1.
  - *MinMaxScaler* shifts the data such that all features are exactly between 0 and 1.
  - *RobustScaler* uses the median and interquartile range to ignore outliers.

a synthetic two-class classification dataset with two features



scikit-learn Practice: *StandardScaler*

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

sklearn.preprocessing.StandardScaler

class `sklearn.preprocessing.` **`StandardScaler`**(*, *copy=True, with_mean=True, with_std=True*) [source]

Standardize features by removing the mean and scaling to unit variance

The standard score of a sample `x` is calculated as:

z = (x - u) / s

where `u` is the mean of the training samples or zero if `with_mean=False`, and `s` is the standard deviation of the training samples or one if `with_std=False`.

Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Mean and standard deviation are then stored to be used on later data using **`transform`**.

Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual features do not more or less look like standard normally distributed data (e.g. Gaussian with 0 mean and unit variance).

For instance many elements used in the objective function of a learning algorithm (such as the RBF kernel of Support Vector Machines or the L1 and L2 regularizers of linear models) assume that all features are centered around 0 and have variance in the same order. If a feature has a variance that is orders of magnitude larger that others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected.

This scaler can also be applied to sparse CSR or CSC matrices by passing `with_mean=False` to avoid breaking the sparsity structure of the data.

scikit-learn Practice: *StandardScaler*

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

Methods

`fit`(X[, y, sample_weight])	Compute the mean and std to be used for later scaling.
`fit_transform`(X[, y])	Fit to data, then transform it.
`get_params`([deep])	Get parameters for this estimator.
`inverse_transform`(X[, copy])	Scale back the data to the original representation
`partial_fit`(X[, y, sample_weight])	Online computation of mean and std on X for later scaling.
`set_params`(**params)	Set the parameters of this estimator.
`transform`(X[, copy])	Perform standardization by centering and scaling

scikit-learn Practice: *StandardScaler*

In [2]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=1)
```

In [3]:
```
pd.DataFrame(X_train).head()
```

	0	1	2	3	4	5	6	7	8	9	...
0	15.22	30.62	103.40	716.9	0.10480	0.20870	0.25500	0.094290	0.2128	0.07152	...
1	14.96	19.10	97.03	687.3	0.08992	0.09823	0.05940	0.048190	0.1879	0.05852	...
2	14.68	20.13	94.74	684.5	0.09867	0.07200	0.07395	0.052590	0.1586	0.05922	...
3	10.32	16.35	65.31	324.9	0.09434	0.04994	0.01012	0.005495	0.1885	0.06201	...
4	11.85	17.46	75.54	432.7	0.08372	0.05642	0.02688	0.022800	0.1875	0.05715	...

In [4]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [5]:
```
pd.DataFrame(X_train_scaled).head()
```

	0	1	2	3	4	5	6	7	8	9	...
0	0.305754	2.595219	0.462461	0.168272	0.604222	2.044178	2.093529	1.163667	1.181984	1.284296	...
1	0.233517	-0.053349	0.205731	0.086315	-0.474245	-0.124576	-0.366220	-0.016409	0.263950	-0.633174	...
2	0.155724	0.183459	0.113437	0.078562	0.159934	-0.639524	-0.183248	0.096223	-0.816308	-0.529925	...
3	-1.055627	-0.685603	-1.072681	-0.917106	-0.153894	-1.072608	-0.985935	-1.109322	0.286071	-0.118407	...
4	-0.630543	-0.430402	-0.660381	-0.618627	-0.923606	-0.945392	-0.775172	-0.666346	0.249202	-0.835245	...

scikit-learn Practice: *MinMaxScaler*

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

sklearn.preprocessing.MinMaxScaler

class sklearn.preprocessing.**MinMaxScaler**(*feature_range=0, 1, *, copy=True, clip=False*) [source]

Transform features by scaling each feature to a given range.

This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g. between zero and one.

The transformation is given by:

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
X_scaled = X_std * (max - min) + min
```

where min, max = feature_range.

This transformation is often used as an alternative to zero mean, unit variance scaling.

Read more in the User Guide.

Parameters:

feature_range : *tuple (min, max), default=(0, 1)*
Desired range of transformed data.

scikit-learn Practice: *MinMaxScaler*

In [2]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=1)
```

In [3]:
```
pd.DataFrame(X_train).head()
```

	0	1	2	3	4	5	6	7	8	9	...
0	15.22	30.62	103.40	716.9	0.10480	0.20870	0.25500	0.094290	0.2128	0.07152	...
1	14.96	19.10	97.03	687.3	0.08992	0.09823	0.05940	0.048190	0.1879	0.05852	...
2	14.68	20.13	94.74	684.5	0.09867	0.07200	0.07395	0.052590	0.1586	0.05922	...
3	10.32	16.35	65.31	324.9	0.09434	0.04994	0.01012	0.005495	0.1885	0.06201	...
4	11.85	17.46	75.54	432.7	0.08372	0.05642	0.02688	0.022800	0.1875	0.05715	...

In [6]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(-1,1))
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [7]:
```
pd.DataFrame(X_train_scaled).head()
```

	0	1	2	3	4	5	6	7	8	9	...
0	-0.220124	0.414271	-0.176145	-0.513552	-0.058048	0.416430	0.194939	-0.062724	0.078788	-0.064821	...
1	-0.244735	-0.364897	-0.264184	-0.538664	-0.326713	-0.410070	-0.721649	-0.520974	-0.172727	-0.636124	...
2	-0.271239	-0.295232	-0.295833	-0.541039	-0.168728	-0.606315	-0.653468	-0.477237	-0.468687	-0.605361	...
3	-0.683942	-0.550896	-0.702578	-0.846108	-0.246908	-0.771360	-0.952577	-0.945378	-0.166667	-0.482751	...
4	-0.539117	-0.475820	-0.561191	-0.754655	-0.438657	-0.722879	-0.874039	-0.773360	-0.176768	-0.696330	...

scikit-learn Practice: *RobustScaler*

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html

sklearn.preprocessing.RobustScaler

class sklearn.preprocessing.**RobustScaler**(*, *with_centering=True*, *with_scaling=True*, *quantile_range=(25.0, 75.0)*, *copy=True*, *unit_variance=False*) [source]

Scale features using statistics that are robust to outliers.

This Scaler removes the median and scales the data according to the quantile range (defaults to IQR: Interquartile Range). The IQR is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile).

Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Median and interquartile range are then stored to be used on later data using the `transform` method.

Standardization of a dataset is a common requirement for many machine learning estimators. Typically this is done by removing the mean and scaling to unit variance. However, outliers can often influence the sample mean / variance in a negative way. In such cases, the median and the interquartile range often give better results.

New in version 0.17.

Read more in the User Guide.

quantile_range : *tuple (q_min, q_max), 0.0 < q_min < q_max < 100.0, default=(25.0, 75.0)*

Quantile range used to calculate `scale_`. By default this is equal to the IQR, i.e., `q_min` is the first quantile and `q_max` is the third quantile.

Effect of Preprocessing on Supervised Learning

- **Example (*breast_cancer* dataset)**

In [8]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=0)
```

In [9]:
```
clf = SVC()
clf.fit(X_train, y_train)
```

Out[9]:
```
▾ SVC
SVC()
```

In [10]:
```
y_train_hat = clf.predict(X_train)
print('train accuracy: %.5f'%accuracy_score(y_train, y_train_hat))

y_test_hat = clf.predict(X_test)
print('test accuracy: %.5f'%accuracy_score(y_test, y_test_hat))
```

```
train accuracy: 0.90376
test accuracy: 0.93706
```

Effect of Preprocessing on Supervised Learning

- **Example (*breast_cancer* dataset)**

In [8]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=0)
```

In [11]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [12]:
```
clf = SVC()
clf.fit(X_train_scaled, y_train)
```

Out[12]:
```
▾ SVC
SVC()
```

In [13]:
```
y_train_hat = clf.predict(X_train_scaled)
print('train accuracy: %.5f'%accuracy_score(y_train, y_train_hat))

y_test_hat = clf.predict(X_test_scaled)
print('test accuracy: %.5f'%accuracy_score(y_test, y_test_hat))
```

```
train accuracy: 0.98592
test accuracy: 0.96503
```

Results without Scaling

```
train accuracy: 0.90376
test accuracy: 0.93706
```

Effect of Preprocessing on Supervised Learning

- **Example (*breast_cancer* dataset)**

In [8]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=0)
```

In [14]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(-1,1))
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [15]:
```
clf = SVC()
clf.fit(X_train_scaled, y_train)
```

Out[15]:
```
▾ SVC
SVC()
```

In [16]:
```
y_train_hat = clf.predict(X_train_scaled)
print('train accuracy: %.5f'%accuracy_score(y_train, y_train_hat))

y_test_hat = clf.predict(X_test_scaled)
print('test accuracy: %.5f'%accuracy_score(y_test, y_test_hat))
```

```
train accuracy: 0.98357
test accuracy: 0.97203
```

Results without Scaling

```
train accuracy: 0.90376
test accuracy: 0.93706
```