Representing Data and Engineering Features

ESM3081 Programming for Data Science

Seokho Kang

성균관대학교
SUNG KYUN KWAN UNIVERSITY(SKKU)

Learning algorithms covered in this course

- **Supervised Learning** (Classification/Regression)

    - K-Nearest Neighbors

    - Linear Models (Logistic/Linear Regression)

    - Decision Trees

    - Random Forests

    - Gradient Boosting Machines

    - Support Vector Machines

    - Neural Networks

Learning algorithms covered in this course

- **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

        - …

Feature Engineering

- The question of how to represent your data best for a particular application is known as *feature engineering*,

  - It is one of the main tasks of data scientists and machine learning practitioners trying to solve real-world problems.

  - How you represent them can have an enormous effect on the performance of machine learning models.

  - While **tree-based models** only **care about the ordering of the features**, **most of the other models** including linear models and neural networks are very **tied to the scale and distribution of each feature**.

  "feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work. feature engineering is fundamental to the application of machine learning, and is both difficult and expensive." (Wikipedia)

  "coming up with features is difficult, time-consuming, requires expert knowledge. applied machine learning is basically feature engineering." (Andrew Ng)

Topics

- **Categorical Features**

- **Binning (Discretization)**

- **Polynomials and Interactions**

- **Univariate Nonlinear Transformations**

- **Automatic Feature Selection**

- **Utilizing Expert Knowledge**

Categorical Features

Categorical Features

- **Example with the *adult* dataset**

    - Adult incomes in the United States, derived from the 1994 census database.

    - The task of the adult dataset is to predict whether a worker has an income of over $50,000 or under $50,000.

    - The input features in this dataset include the workers' ages, how they are employed (self employed, private industry employee, government employee, etc.), their education, their gender, their working hours per week, occupation, and more.

Categorical Features

- **Example with the *adult* dataset**

  - *age* and *hours-per-week* are **continuous features**, which we know how to treat.

  - *workclass*, *education*, *gender,* and *occupation* are **categorical features**. They come from a fixed list of possible values.

  - *The first few entries in the adult dataset*

	age	workclass	education	gender	hours-per-week	occupation	income
0	39	State-gov	Bachelors	Male	40	Adm-clerical	<=50K
1	50	Self-emp-not-inc	Bachelors	Male	13	Exec-managerial	<=50K
2	38	Private	HS-grad	Male	40	Handlers-cleaners	<=50K
3	53	Private	11th	Male	40	Handlers-cleaners	<=50K
4	28	Private	Bachelors	Female	40	Prof-specialty	<=50K
5	37	Private	Masters	Female	40	Exec-managerial	<=50K
6	49	Private	9th	Female	16	Other-service	<=50K
7	52	Self-emp-not-inc	HS-grad	Male	45	Exec-managerial	>50K
8	31	Private	Masters	Female	50	Prof-specialty	>50K
9	42	Private	Bachelors	Male	40	Exec-managerial	>50K
10	37	Private	Some-college	Male	80	Exec-managerial	>50K

pandas Practice

- **Example (*adult* dataset)**
  - We load the data using pandas from a comma-separated values (CSV) file

```
In [2]:  import mglearn, os

         adult_path = os.path.join(mglearn.datasets.DATA_PATH, 'adult.data')
         print(adult_path)

C:\ProgramData\Anaconda3\lib\site-packages\mglearn\data\adult.data
```

```
 1   39, State-gov, 77516, Bachelors, 13, Never-married, Adm-clerical, Not-in-family, White, Male, 2174, 0, 40, United-States, <=50K
 2   50, Self-emp-not-inc, 83311, Bachelors, 13, Married-civ-spouse, Exec-managerial, Husband, White, Male, 0, 0, 13, United-States, <=50K
 3   38, Private, 215646, HS-grad, 9, Divorced, Handlers-cleaners, Not-in-family, White, Male, 0, 0, 40, United-States, <=50K
 4   53, Private, 234721, 11th, 7, Married-civ-spouse, Handlers-cleaners, Husband, Black, Male, 0, 0, 40, United-States, <=50K
 5   28, Private, 338409, Bachelors, 13, Married-civ-spouse, Prof-specialty, Wife, Black, Female, 0, 0, 40, Cuba, <=50K
 6   37, Private, 284582, Masters, 14, Married-civ-spouse, Exec-managerial, Wife, White, Female, 0, 0, 40, United-States, <=50K
 7   49, Private, 160187, 9th, 5, Married-spouse-absent, Other-service, Not-in-family, Black, Female, 0, 0, 16, Jamaica, <=50K
 8   52, Self-emp-not-inc, 209642, HS-grad, 9, Married-civ-spouse, Exec-managerial, Husband, White, Male, 0, 0, 45, United-States, >50K
 9   31, Private, 45781, Masters, 14, Never-married, Prof-specialty, Not-in-family, White, Female, 14084, 0, 50, United-States, >50K
10   42, Private, 159449, Bachelors, 13, Married-civ-spouse, Exec-managerial, Husband, White, Male, 5178, 0, 40, United-States, >50K
11   37, Private, 280464, Some-college, 10, Married-civ-spouse, Exec-managerial, Husband, Black, Male, 0, 0, 80, United-States, >50K
12   30, State-gov, 141297, Bachelors, 13, Married-civ-spouse, Prof-specialty, Husband, Asian-Pac-Islander, Male, 0, 0, 40, India, >50K
13   23, Private, 122272, Bachelors, 13, Never-married, Adm-clerical, Own-child, White, Female, 0, 0, 30, United-States, <=50K
14   32, Private, 205019, Assoc-acdm, 12, Never-married, Sales, Not-in-family, Black, Male, 0, 0, 50, United-States, <=50K
15   40, Private, 121772, Assoc-voc, 11, Married-civ-spouse, Craft-repair, Husband, Asian-Pac-Islander, Male, 0, 0, 40, ?, >50K
16   34, Private, 245487, 7th-8th, 4, Married-civ-spouse, Transport-moving, Husband, Amer-Indian-Eskimo, Male, 0, 0, 45, Mexico, <=50K
17   25, Self-emp-not-inc, 176756, HS-grad, 9, Never-married, Farming-fishing, Own-child, White, Male, 0, 0, 35, United-States, <=50K
18   32, Private, 186824, HS-grad, 9, Never-married, Machine-op-inspct, Unmarried, White, Male, 0, 0, 40, United-States, <=50K
19   38, Private, 28887, 11th, 7, Married-civ-spouse, Sales, Husband, White, Male, 0, 0, 50, United-States, <=50K
20   43, Self-emp-not-inc, 292175, Masters, 14, Divorced, Exec-managerial, Unmarried, White, Female, 0, 0, 45, United-States, >50K
21   40, Private, 193524, Doctorate, 16, Married-civ-spouse, Prof-specialty, Husband, White, Male, 0, 0, 60, United-States, >50K
22   54, Private, 302146, HS-grad, 9, Separated, Other-service, Unmarried, Black, Female, 0, 0, 20, United-States, <=50K
23   35, Federal-gov, 76845, 9th, 5, Married-civ-spouse, Farming-fishing, Husband, Black, Male, 0, 0, 40, United-States, <=50K
24   43, Private, 117037, 11th, 7, Married-civ-spouse, Transport-moving, Husband, White, Male, 0, 2042, 40, United-States, <=50K
25   59, Private, 109015, HS-grad, 9, Divorced, Tech-support, Unmarried, White, Female, 0, 0, 40, United-States, <=50K
26   56, Local-gov, 216851, Bachelors, 13, Married-civ-spouse, Tech-support, Husband, White, Male, 0, 0, 40, United-States, >50K
27   19, Private, 168294, HS-grad, 9, Never-married, Craft-repair, Own-child, White, Male, 0, 0, 40, United-States, <=50K
28   54, ?, 180211, Some-college, 10, Married-civ-spouse, ?, Husband, Asian-Pac-Islander, Male, 0, 0, 60, South, >50K
29   39, Private, 367260, HS-grad, 9, Divorced, Exec-managerial, Not-in-family, White, Male, 0, 0, 80, United-States, <=50K
30   49, Private, 193366, HS-grad, 9, Married-civ-spouse, Craft-repair, Husband, White, Male, 0, 0, 40, United-States, <=50K
31   23, Local-gov, 190709, Assoc-acdm, 12, Never-married, Protective-serv, Not-in-family, White, Male, 0, 0, 52, United-States, <=50K
32   20, Private, 266015, Some-college, 10, Never-married, Sales, Own-child, Black, Male, 0, 0, 44, United-States, <=50K
33   45, Private, 386940, Bachelors, 13, Divorced, Exec-managerial, Own-child, White, Male, 0, 1408, 40, United-States, <=50K
```

pandas Practice

- **Example (*adult* dataset)**
  - The table below shows the first five rows of the adult dataset

The file has no headers naming the columns, so we pass header=None and provide the column names explicitly in "names"

```
In [3]:  import pandas as pd

         data = pd.read_csv(
             adult_path, header=None, index_col=False,
             names=['age', 'workclass', 'fnlwgt', 'education', 'education-num',
                    'marital-status', 'occupation', 'relationship', 'race', 'gender',
                    'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',
                    'income'])

         display(data.head())
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country	income
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

pandas Practice

- **Example (*adult* dataset)**
  - For illustration purposes, we only select some of the columns.
    - * *age* and *hours-per-week* are **continuous features**
    - * *workclass*, *education*, *sex*, and *occupation* are **categorical features**

```
In [4]: data = data[['age', 'workclass', 'education', 'gender', 'hours-per-week','occupation', 'income']]

display(data.head())
```

	age	workclass	education	gender	hours-per-week	occupation	income
0	39	State-gov	Bachelors	Male	40	Adm-clerical	<=50K
1	50	Self-emp-not-inc	Bachelors	Male	13	Exec-managerial	<=50K
2	38	Private	HS-grad	Male	40	Handlers-cleaners	<=50K
3	53	Private	11th	Male	40	Handlers-cleaners	<=50K
4	28	Private	Bachelors	Female	40	Prof-specialty	<=50K

pandas Practice

- **Example (*adult* dataset)**
  - It is often good to first check if a column actually contains meaningful categorical data.
  - A good way to check the contents of a column is using the *value_counts* method of a pandas to show us what the unique values are and how often they appear:

```
In [4]: print(data.income.value_counts())

        <=50K    24720
        >50K      7841
        Name: income, dtype: int64
```

```
In [5]: print(data.gender.value_counts())

        Male      21790
        Female    10771
        Name: gender, dtype: int64
```

```
In [6]: print(data.workclass.value_counts())

        Private            22696
        Self-emp-not-inc    2541
        Local-gov           2093
        ?                   1836
        State-gov           1298
        Self-emp-inc        1116
        Federal-gov          960
        Without-pay           14
        Never-worked           7
        Name: workclass, dtype: int64
```

One-Hot Encoding

- When applying some learning algorithms (*e.g.*, k-NN, linear models, SVM, neural networks), categorical features must be transformed into numeric features.

- The most common way to represent categorical features is using the ***one-hot encoding*** *(a.k.a., dummy features)*.

  - Replace a categorical feature with multiple new features that can have the values 0 and 1 (one new feature per category)

  - ***Example:*** *encoding the workclass feature using one-hot encoding*

ID	Workclass
1	Government Employee
2	Private Employee
3	Self Employed
4	Self Employed
5	Self Employed Incorporated
6	Private Employee
7	Self Employed
8	Self Employed
9	Private Employee
10	Government Employee

One-hot encoding →
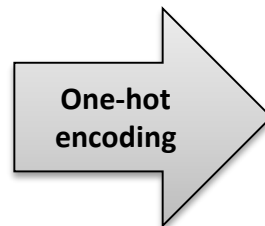
ID	Workclass_ Government Employee	Workclass_ Private Employee	Workclass_ Self Employed	Workclass_ Self Employed Incorporated
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	1	0
5	0	0	0	1
6	0	1	0	0
7	0	0	1	0
8	0	0	1	0
9	0	1	0	0
10	1	0	0	0

pandas Practice

- **Example (*adult* dataset)**
  - The *get_dummies* function in pandas automatically transforms all columns that have object type (like strings) or are categorical.
  - The continuous features age and hours-per-week were not touched, while the categorical features were expanded into one new feature for each possible value.

```
In [7]: data_dummies = pd.get_dummies(data)
        display(data_dummies.head())
```

	age	hours-per-week	workclass_?	workclass_Federal-gov	workclass_Local-gov	workclass_Never-worked	workclass_Private	workclass_Self-emp-inc	workclass_Self-emp-not-inc	workclass_State-gov	...
0	39	40	0	0	0	0	0	0	0	1	...
1	50	13	0	0	0	0	0	0	1	0	...
2	38	40	0	0	0	0	1	0	0	0	...
3	53	40	0	0	0	0	1	0	0	0	...
4	28	40	0	0	0	0	1	0	0	0	...

5 rows × 46 columns

```
In [8]: X = data_dummies.loc[:, 'age':'occupation_ Transport-moving'].values
        y = data_dummies['income_ >50K'].values

        print("X.shape: {} y.shape: {}".format(X.shape, y.shape))

        X.shape: (32561, 44) y.shape: (32561,)
```

pandas Practice

- **Example (*adult* dataset)**

  - Categorical features are often encoded using integers. That they are numbers doesn't mean that they should necessarily be treated as continuous features.

  - If you want dummy features to be created for the "integer feature" column, you can explicitly list the columns you want to encode using the columns parameter.

```
In [9]: data_dummies = pd.get_dummies(data, columns=['age','workclass'])
        display(data_dummies.head())
```

	education	gender	hours-per-week	occupation	income	age_17	age_18	age_19	age_20	age_21	...
0	Bachelors	Male	40	Adm-clerical	<=50K	0	0	0	0	0	...
1	Bachelors	Male	13	Exec-managerial	<=50K	0	0	0	0	0	...
2	HS-grad	Male	40	Handlers-cleaners	<=50K	0	0	0	0	0	...
3	11th	Male	40	Handlers-cleaners	<=50K	0	0	0	0	0	...
4	Bachelors	Female	40	Prof-specialty	<=50K	0	0	0	0	0	...

5 rows × 87 columns

Binning (Discretization)

Binning (Discretization)

- The best way to represent data depends not only on the semantics of the data, but also on the kind of model you are using.

    - Linear models can only model linear relationships.

    - Decision trees can build a much more complex model of the data.

    - *Example: Comparing linear regression and a decision tree on the wave dataset*

- One way to make linear models more powerful on continuous data is to use *binning* (*discretization*) of the feature to split it up into multiple features.
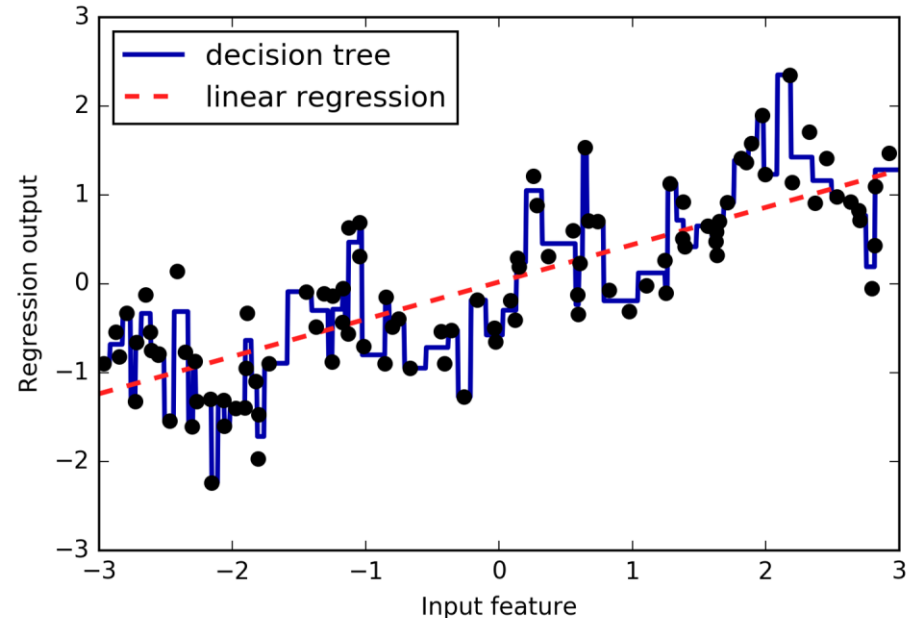
scikit-learn Practice

- **Example (*wave* dataset)**

```
In [10]: import mglearn

         X, y = mglearn.datasets.make_wave(n_samples=100)
         print(X[:5])

         [[-0.75275929]
          [ 2.70428584]
          [ 1.39196365]
          [ 0.59195091]
          [-2.06388816]]
```

We partition the input range for the feature into a fixed number of bins.
Then, we transform the single continuous input feature in the wave dataset into a categorical feature that encodes which bin a data point is in.

```
In [11]: import numpy as np

         bins = np.linspace(-3, 3, 11)
         print("bins: {}".format(bins))
         which_bin = np.digitize(X, bins=bins)

         bins: [-3.  -2.4 -1.8 -1.2 -0.6  0.   0.6  1.2  1.8  2.4  3. ]
```

```
In [12]: from sklearn.preprocessing import OneHotEncoder

         encoder = OneHotEncoder(sparse_output=False)
         encoder.fit(which_bin)
         X_binned = encoder.transform(which_bin)
         print(X_binned[:5])

         [[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
          [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
          [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
          [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
          [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

We transform this discrete feature to a one-hot encoding.

18

scikit-learn Practice

- **Example (*wave* dataset)**
  - The linear model became much more flexible, because it now has a different value for each bin.
  - The decision tree model got much less flexible.

```
In [13]:  from sklearn.linear_model import LinearRegression
          reg_lr = LinearRegression().fit(X, y)
          reg_lr_bin = LinearRegression().fit(X_binned, y)

          from sklearn.tree import DecisionTreeRegressor
          reg_dt = DecisionTreeRegressor(min_samples_split=3).fit(X, y)
          reg_dt_bin = DecisionTreeRegressor(min_samples_split=3).fit(X_binned, y)
```
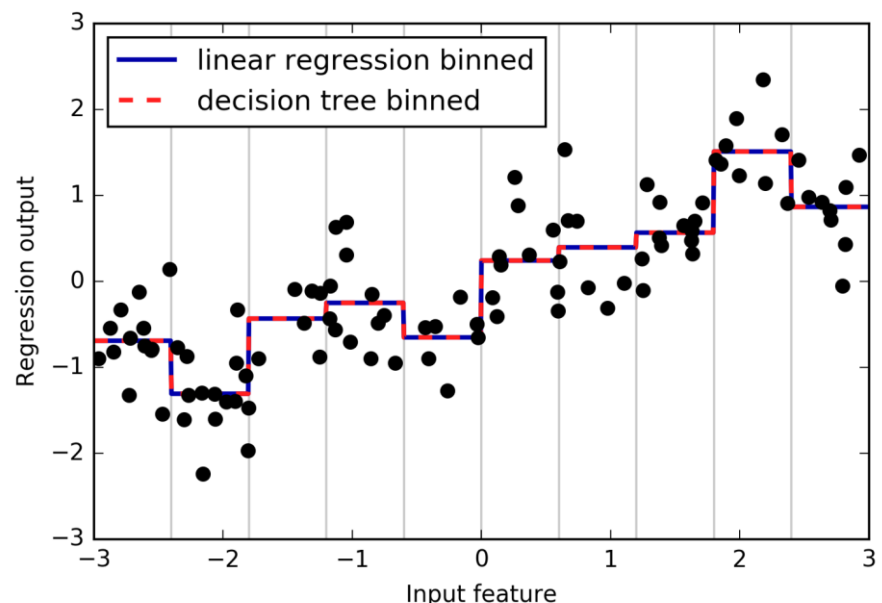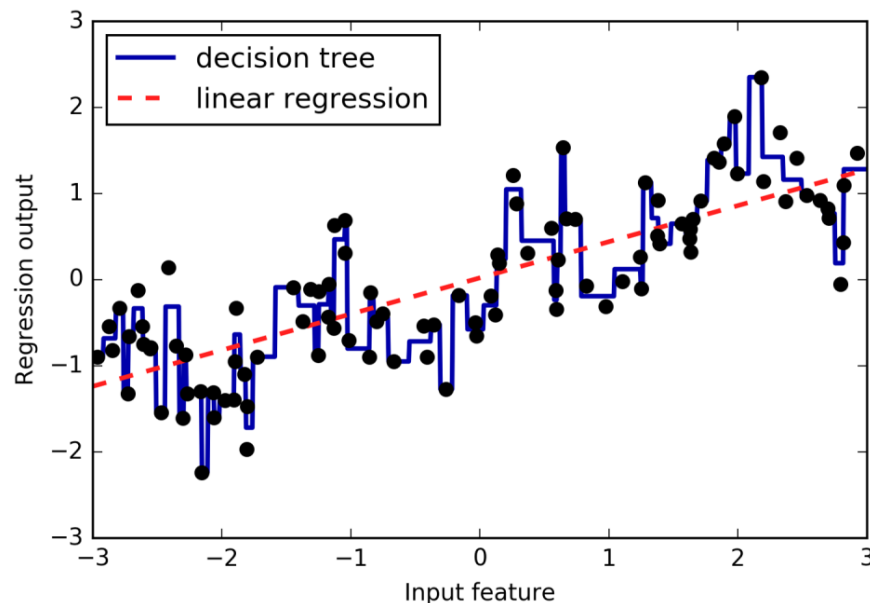
Discussion

- **The linear model benefits greatly in expressiveness from *binning* of continuous features.**
  - If there are good reasons to use a linear model but some features have nonlinear relations with the output, binning can be a great way to increase modeling power.

- **Binning features generally has no beneficial effect for decision trees.**
  - Decision trees can learn to split up the data anywhere and learn whatever binning is most useful for predicting on this data.
  - Additionally, decision trees look at multiple features at once, while binning is usually done on a per-feature basis.

- **Nonlinear models might be able to learn more complex tasks without using binning.**

Polynomials and Interactions

Polynomials

- **For linear models, one way to enrich a feature representation is adding *Polynomial Features* of the original data.**

- **Example: Univariate Polynomial Regression**

  - A linear regression model (polynomial degree=1)
  $$\hat{y} = w_0 + w_1 x$$

  - A quadratic regression model (polynomial degree=2)
  $$\hat{y} = w_0 + w_1 x + w_2 x^2$$

  x^2: square of x

  - A polynomial regression model (polynomial degree=9)

  $$\hat{y} = w_0 + \sum_{i=1}^{9} w_i x^i$$

  x^i: i-th power of x

Underfitting · Appropriate capacity · Overfitting
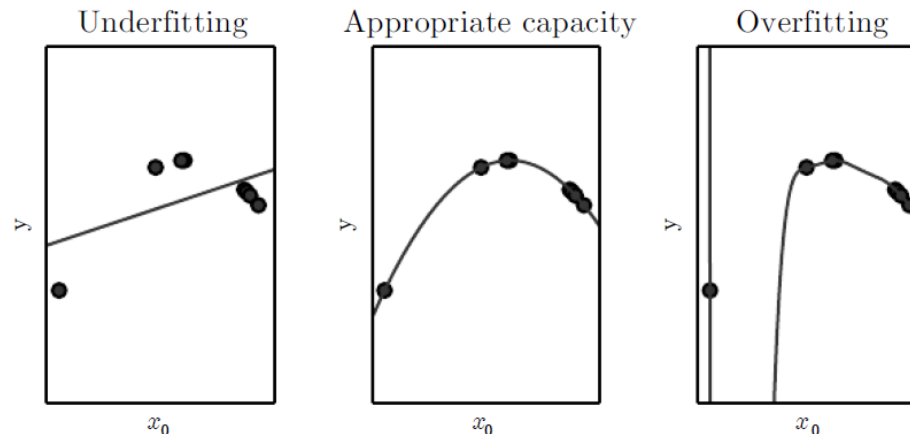
Interactions

- *Interaction Features* – products between original features

  - $(x_1, x_2) \rightarrow (x_1 x_2)$

  - $(x_1, x_2, x_3) \rightarrow (x_1 x_2, x_2 x_3, x_1 x_3, x_1 x_2 x_3)$

  - …

- **Example: Bivariate Quadratic Regression with Interaction**

$$\hat{y} = w_{00} + w_{10} x_1 + w_{01} x_2 + w_{20} x_1^2 + w_{11} x_1 x_2 + w_{02} x_2^2$$

scikit-learn Practice

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

sklearn.preprocessing.PolynomialFeatures

class sklearn.preprocessing.**PolynomialFeatures**(*degree=2, *, interaction_only=False, include_bias=True, order='C'*)   [source]

Generate polynomial and interaction features.

Generate a new feature matrix consisting of all polynomial combinations of the features with degree less than or equal to the specified degree. For example, if an input sample is two dimensional and of the form [a, b], the degree-2 polynomial features are [1, a, b, a^2, ab, b^2].

Parameters:	**degree** : *int, default=2* The degree of the polynomial features. **interaction_only** : *bool, default=False* If true, only interaction features are produced: features that are products of at most `degree` *distinct* input features (so not `x[1] ** 2`, `x[0] * x[2] ** 3`, etc.). **include_bias** : *bool, default=True* If True (default), then include a bias column, the feature in which all polynomial powers are zero (i.e. a column of ones - acts as an intercept term in a linear model).

scikit-learn Practice

- **Example (*wave* dataset)**

```
In [14]:  import mglearn
          X, y = mglearn.datasets.make_wave(n_samples=100)
          print(X[:5])

          [[-0.75275929]
           [ 2.70428584]
           [ 1.39196365]
           [ 0.59195091]
           [-2.06388816]]
```

```
In [15]:  from sklearn.preprocessing import PolynomialFeatures
          poly = PolynomialFeatures(degree=3, include_bias=False)
          poly.fit(X)
          poly.get_feature_names_out()

Out[15]:  ['x0', 'x0^2', 'x0^3']
```

```
In [16]:  X_poly = poly.transform(X)
          print(X_poly[:5])

          [[-0.75275929   0.56664654  -0.42654845]
           [ 2.70428584   7.3131619   19.77688015]
           [ 1.39196365   1.93756281   2.697017  ]
           [ 0.59195091   0.35040587   0.20742307]
           [-2.06388816   4.25963433  -8.79140884]]
```

scikit-learn Practice

- **Example (*wave* dataset)**

  - Polynomial features yield a very smooth fit on this one-dimensional data.

  - Polynomials of very high degree tend to behave in extreme ways on the boundaries or in regions with little data. → *overfitting*
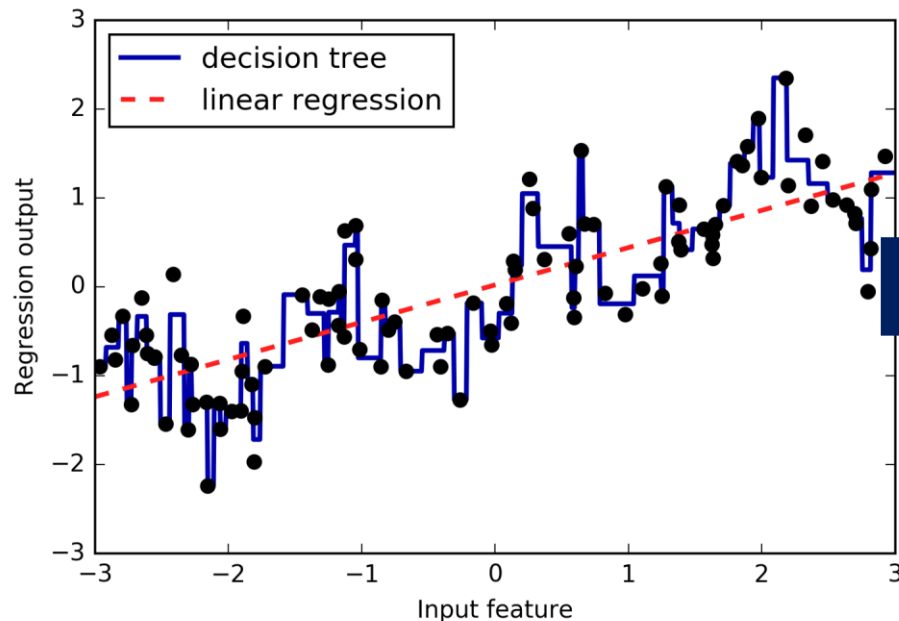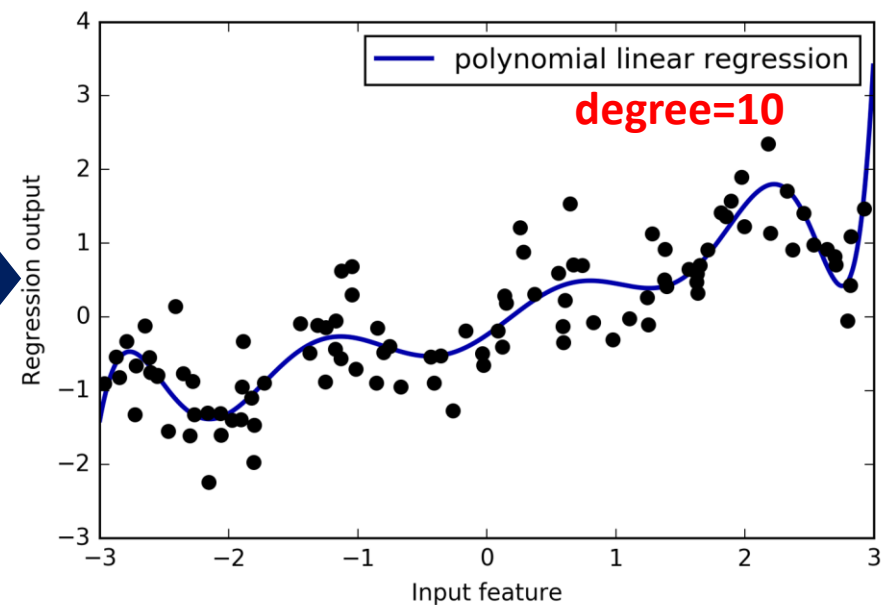
scikit-learn Practice

- **Example (*iris* dataset) without polynomial/interaction features**

```
In [18]: from sklearn.datasets import load_iris
         from sklearn.model_selection import train_test_split
         from sklearn.preprocessing import StandardScaler
         from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import accuracy_score

         iris = load_iris()
         X_train, X_test, y_train, y_test = train_test_split(
             iris.data, iris.target, random_state=0)
```

```
In [19]: scaler = StandardScaler()
         scaler.fit(X_train)
         X_train_scaled = scaler.transform(X_train)
         X_test_scaled = scaler.transform(X_test)
```

```
In [20]: clf = LogisticRegression()
         clf.fit(X_train_scaled, y_train)
```

```
Out[20]:  ▼ LogisticRegression
          LogisticRegression()
```

```
In [21]: 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: 0.97321
test accuracy: 0.97368
```

The performance using logistic regression on the data without polynomial features

27

scikit-learn Practice

- **Example (*iris* dataset) with polynomial/interaction features**

```
In [22]: from sklearn.datasets import load_iris
         from sklearn.model_selection import train_test_split
         from sklearn.preprocessing import PolynomialFeatures
         from sklearn.preprocessing import StandardScaler
         from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import accuracy_score

         iris = load_iris()
         X_train, X_test, y_train, y_test = train_test_split(
             iris.data, iris.target, random_state=42)
```

The data originally have 4 features

```
In [23]: poly = PolynomialFeatures(degree=2, include_bias=False)
         poly.fit(X_train)
         poly.get_feature_names_out()
```
```
Out[23]: ['x0',
          'x1',
          'x2',
          'x3',
          'x0^2',
          'x0 x1',
          'x0 x2',
          'x0 x3',
          'x1^2',
          'x1 x2',
          'x1 x3',
          'x2^2',
          'x2 x3',
          'x3^2']
```

New features represent all possible interactions between two different original features, as well as the square of each original feature.

```
In [24]: X_train_poly = poly.transform(X_train)
         X_test_poly = poly.transform(X_test)
         print(X_train.shape, X_train_poly.shape)

         (112, 4) (112, 14)
```

28

scikit-learn Practice

- **Example (*iris* dataset) with polynomial/interaction features**

```
In [25]: scaler = StandardScaler()
         scaler.fit(X_train_poly)
         X_train_poly_scaled = scaler.transform(X_train_poly)
         X_test_poly_scaled = scaler.transform(X_test_poly)
```

```
In [26]: clf = LogisticRegression()
         clf.fit(X_train_poly_scaled, y_train)
```

```
Out[26]:     ▼ LogisticRegression
         LogisticRegression()
```

```
In [27]: y_train_hat = clf.predict(X_train_poly_scaled)
         print('train accuracy: %.5f'%accuracy_score(y_train, y_train_hat))

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

```
train accuracy: 0.97321
test accuracy: 1.00000
```

The performance using logistic regression on the data with polynomial features

Discussion

- **The polynomial and interaction features give us a good boost in performance when using linear models.**

- **Nonlinear models might be able to learn more complex tasks without using polynomial and interaction features.**

Univariate Nonlinear Transformations

Univariate Nonlinear Transformations

- **Most models work best when each feature (and in regression also the target) is loosely Gaussian distributed.**

    - A histogram of each feature should have something resembling the familiar "bell curve" shape.

    - Using transformations like log and exp is a hacky but simple and efficient way to achieve this.

- **Univariate non-linear transformations often prove useful for transforming certain features.**

    - The functions log and exp can help by adjusting the relative scales in the data so that they can be captured better by a linear model or neural network.

Univariate Nonlinear Transformations

- **Example: Logarithmic Transformation**

  - **Histogram of original feature values X**

    - The counts for higher values fall quickly.

    - This kind of distribution of values (many small ones and a few very large ones) is very common in practice.

  - **Histogram of transformed feature values log(X+1)**

    - After the transformation, the distribution of the data is less asymmetrical and doesn't have very large outliers anymore.
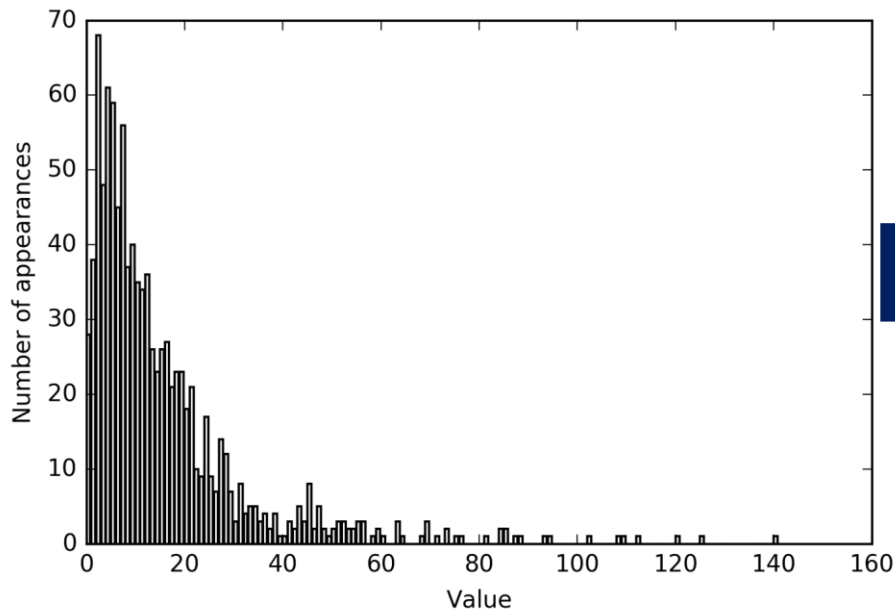
$\log(X + 1)$
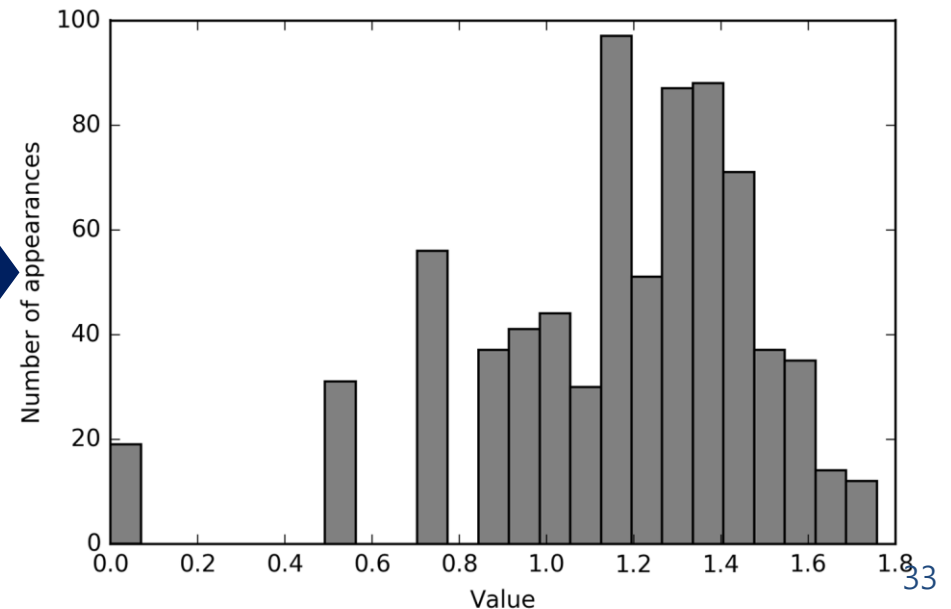
numpy Practice

numpy.log

numpy.log(x, /, out=None, *, where=True, casting='same_kind', order='K', dtype=None, subok=True[, signature, extobj]) = <ufunc 'log'>

> Natural logarithm, element-wise.
>
> The natural logarithm **log** is the inverse of the exponential function, so that *log(exp(x)) = x*. The natural logarithm is logarithm in base **e**.

numpy.exp

numpy.exp(x, /, out=None, *, where=True, casting='same_kind', order='K', dtype=None, subok=True[, signature, extobj]) = <ufunc 'exp'>

> Calculate the exponential of all elements in the input array.

- **Example**

```
In [27]:  import numpy as np

          X_train_log = np.log(X_train + 1)
          X_test_log = np.log(X_test + 1)

          X_train_rec = np.exp(X_train_log) − 1
          X_test_rec = np.exp(X_test_log) − 1
```

Discussion

- **Finding the transformation that works best for each combination of dataset and model is somewhat of an art.**

  - Each feature needs to be transformed in a different way.

  - When a feature has a right-skewed distribution (*e.g.*, integer count data), using the log(y + 1) transformation often helps.

- **Univariate nonlinear transformations of features are irrelevant for tree-based models.**

- **Sometimes it is also a good idea to transform the target y in regression.**

Automatic Feature Selection

Overview

- With so many ways to create new features, you might get tempted to increase the dimensionality of the data way beyond the number of original features.

- However, adding more features makes all models more complex, and so increases the chance of overfitting.

- It can be a good idea to reduce the number of features to only the most useful ones, and discard the rest.

    - This can lead to simpler models that generalize better.

    - We need to split the data into training and test sets, and fit the feature selection only on the training part of the data.

Overview

- **Feature Extraction vs Feature Selection**

    - **Feature Extraction** is to extract new features as a combination of all the original features (e.g., PCA)

    - **Feature Selection** is to select a subset of existing features without a transformation

- **When do we prefer "Feature Selection"?**

    - Features are expensive to obtain

    - We want to preserve the original representation of the features (interpretability)

Overview

- **Feature Selection Strategies**

    - **Filter Methods** select features as a pre-processing step, independently of the chosen learning algorithm.

    - **Wrapper Methods** assess subsets of features according to their predictive power to the learning algorithm of interest.

    - **Embedded Methods** perform feature selection in the process of training and are usually specific to given learning algorithms.

        - **e.g.,** Lasso Regression, Decision Trees, Random Forests, …
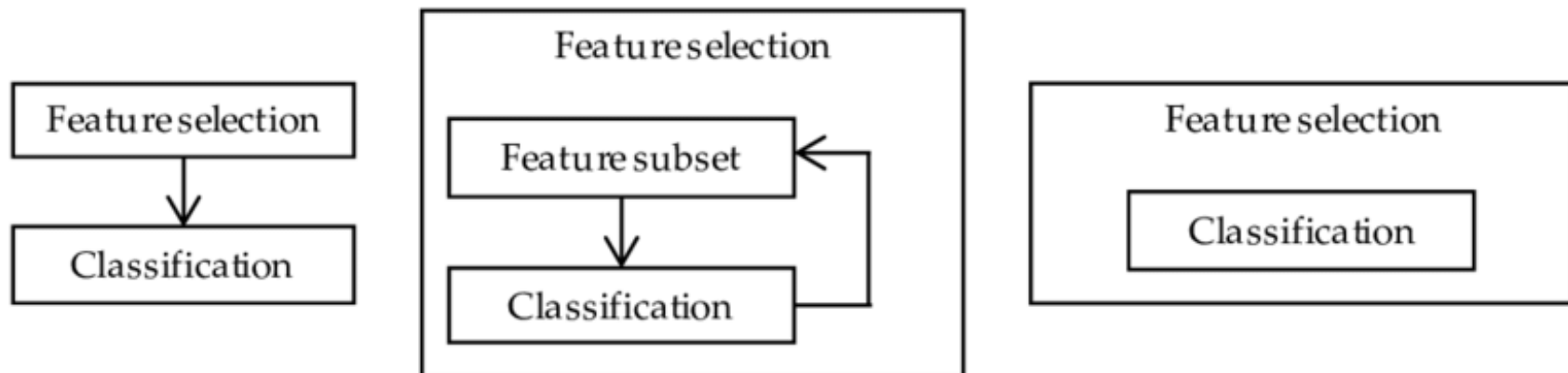
Filter Methods

- **Filter Methods** select features as a pre-processing step, independently of the chosen learning algorithm.

    - We compute whether there is a statistically significant relationship between each feature and the target based on univariate statistics.

    - Then the features that are related with the highest confidence are selected.

    - Consequently, a feature will be discarded if it is only informative when combined with another feature.

    - Univariate tests are often very fast to compute, and don't require building a model.

    - On the other hand, they are completely independent of the model that you might want to apply after the feature selection.

scikit-learn Practice

https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html

sklearn.feature_selection.SelectKBest

class sklearn.feature_selection. **SelectKBest**(*score_func=<function f_classif>, *, k=10*) [source]

Select features according to the k highest scores.

Read more in the User Guide.

Parameters:
score_func : callable, default=f_classif
   Function taking two arrays X and y, and returning a pair of arrays (scores, pvalues) or a single array with scores. Default is f_classif (see below "See Also"). The default function only works with classification tasks.

   New in version 0.18.

k : int or "all", default=10
   Number of top features to select. The "all" option bypasses selection, for use in a parameter search.

Attributes:
scores_ : array-like of shape (n_features,)
   Scores of features.

pvalues_ : array-like of shape (n_features,)
   p-values of feature scores, None if `score_func` returned only scores.

f_classif
   ANOVA F-value between label/feature for classification tasks.
mutual_info_classif
   Mutual information for a discrete target.
chi2
   Chi-squared stats of non-negative features for classification tasks.
f_regression
   F-value between label/feature for regression tasks.
mutual_info_regression
   Mutual information for a continuous target.

scikit-learn Practice

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

```
In [29]: from sklearn.datasets import load_breast_cancer
         from sklearn.preprocessing import StandardScaler
         from sklearn.model_selection import train_test_split
         from sklearn.neighbors import KNeighborsClassifier
         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, stratify=cancer.target, random_state=66)

         scaler = StandardScaler()
         scaler.fit(X_train)

         X_train_scaled = scaler.transform(X_train)
         X_test_scaled = scaler.transform(X_test)
         X_train.shape

Out[29]: (426, 30)

In [30]: clf = KNeighborsClassifier()
         clf.fit(X_train_scaled, y_train)

Out[30]:   ▼ KNeighborsClassifier
         KNeighborsClassifier()

In [31]: 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: 0.98122
test accuracy: 0.95804
```
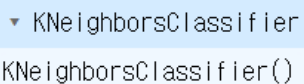
scikit-learn Practice

- **Example (*breast_cancer* dataset)**
  - Feature Importance (Scores of Features)
    - *score_func: f_classif (default)*

```
In [31]: from sklearn.feature_selection import SelectKBest, f_classif

         select = SelectKBest(f_classif, k=20)
         select.fit(X_train_scaled, y_train)
         print(select.scores_)
```

```
[4.83462863e+02 7.30185143e+01 5.28179103e+02 4.47936141e+02
 8.31684872e+01 2.82587643e+02 4.03951190e+02 6.77339474e+02
 6.14526934e+01 6.01858277e-01 2.31866351e+02 1.04667244e+00
 2.34005578e+02 2.98127849e+02 1.51979304e+00 4.25157374e+01
 2.41668418e+01 7.54074868e+01 4.75614984e-03 2.31919142e+00
 6.53070215e+02 1.08039666e+02 7.04457051e+02 5.24963982e+02
 1.37073609e+02 2.95079487e+02 3.73197273e+02 7.97341589e+02
 1.26635084e+02 7.49205074e+01]
```

   - *score_func: mutual_info_classif*

```
In [32]: from sklearn.feature_selection import SelectKBest, mutual_info_classif

         select = SelectKBest(mutual_info_classif, k=20)
         select.fit(X_train_scaled, y_train)
         print(select.scores_)
```

```
[0.37811681 0.05944512 0.3931308  0.36187796 0.07105061 0.2548852
 0.39395125 0.43071963 0.06399398 0.00759013 0.24025362 0.
 0.24203242 0.32941177 0.         0.07178491 0.15369855 0.09221125
 0.02422184 0.04810114 0.46294588 0.11370017 0.4907847  0.46998599
 0.11772397 0.28481903 0.32519726 0.46213516 0.14345567 0.09099679]
```
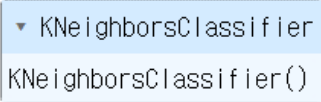
scikit-learn Practice

- **Example (*breast_cancer* dataset) with feature selection** *(score_func: f_classif)*

```
In [34]: from sklearn.feature_selection import SelectKBest, f_classif

         select = SelectKBest(f_classif, k=20)
         select.fit(X_train_scaled, y_train)
         X_train_selected = select.transform(X_train_scaled)
         X_test_selected = select.transform(X_test_scaled)
```

```
In [35]: clf = KNeighborsClassifier()
         clf.fit(X_train_selected, y_train)
```
```
Out[35]:   ▼ KNeighborsClassifier

         KNeighborsClassifier()
```

```
In [36]: y_train_hat = clf.predict(X_train_selected)
         print('train accuracy: %.5f'%accuracy_score(y_train, y_train_hat))

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

         train accuracy: 0.98826
         test accuracy: 0.97902
```

Wrapper Methods

- **Wrapper Methods** assess subsets of features according to their predictive power to the learning algorithm of interest.

  - A series of models are built with varying numbers of features.

  - As wrapper methods allow interactions between features and model dependency to be considered, they usually provide higher prediction accuracy than filter methods.

  - They are much more computationally expensive than filter/embedded methods

Wrapper Methods

- **Exhaustive search:** evaluates all possible combinations of features (2^d-1) → infeasible

- **Sequential methods:** sequentially add or remove features to maximize the prediction accuracy.

  - **Forward selection**: starting with no features and adding features one by one until some stopping criterion is reached

  - **Backward selection**: starting with all features and removing features one by one until some stopping criterion is reached.

  - **Stepwise selection:** alternating between forward and backward, allowing backtracking for the inclusion and exclusion of features

- **Stochastic methods:** incorporate randomness in the search procedure to escape local optima.

  - They typically utilize **metaheuristic optimization algorithms** such as genetic algorithm (GA), particle swarm optimization, ant colony optimization, and simulated annealing.

scikit-learn Practice

https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html

sklearn.feature_selection.RFE

class sklearn.feature_selection. RFE(*estimator, *, n_features_to_select=None, step=1, verbose=0*)   [source]

Feature ranking with recursive feature elimination.

Given an external estimator that assigns weights to features (e.g., the coefficients of a linear model), the goal of recursive feature elimination (RFE) is to select features by recursively considering smaller and smaller sets of features. First, the estimator is trained on the initial set of features and the importance of each feature is obtained either through a `coef_` attribute or through a `feature_importances_` attribute. Then, the least important features are pruned from current set of features. That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached.

Read more in the User Guide.

Parameters:	**estimator : *object***
	A supervised learning estimator with a `fit` method that provides information about feature importance either through a `coef_` attribute or through a `feature_importances_` attribute.
	n_features_to_select : *int or None (default=None)*
	The number of features to select. If `None`, half of the features are selected.
	step : *int or float, optional (default=1)*
	If greater than or equal to 1, then `step` corresponds to the (integer) number of features to remove at each iteration. If within (0.0, 1.0), then `step` corresponds to the percentage (rounded down) of features to remove at each iteration.

scikit-learn Practice

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

```
In [36]: from sklearn.feature_selection import RFE
         from sklearn.svm import SVC

         estimator = SVC(kernel="linear")
         select = RFE(estimator, n_features_to_select=10, step=1)
         select.fit(X_train_scaled, y_train)

         X_train_selected = select.transform(X_train_scaled)
         X_test_selected = select.transform(X_test_scaled)
         select.get_support()
```

```
Out[36]: array([False, False, False, False, False, False, False,  True, False,
                  True, False, False, False,  True, False, False, False, False,
                 False, False,  True,  True,  True,  True,  True, False,  True,
                 False,  True, False])
```

```
In [37]: from sklearn.feature_selection import RFE
         from sklearn.svm import SVC

         estimator = SVC(kernel="linear")
         select = RFE(estimator, n_features_to_select=20, step=1)
         select.fit(X_train_scaled, y_train)

         X_train_selected = select.transform(X_train_scaled)
         X_test_selected = select.transform(X_test_scaled)
         select.get_support()
```

```
Out[37]: array([False, False,  True, False, False, False,  True,  True, False,
                  True,  True,  True, False,  True,  True,  True,  True, False,
                 False,  True,  True,  True,  True,  True,  True, False,  True,
                  True,  True,  True])
```

Embedded Methods

- **Embedded Methods** perform feature selection in the process of training and are usually specific to given learning algorithms.

  - They use a supervised machine learning model to judge the importance of each feature, and keeps only the most important ones.

  - The model needs to provide some measure of importance for each feature, so that they can be ranked by this measure.

    - Decision trees and decision tree–based models provide a *feature_importances_* attribute, which directly encodes the importance of each feature.

    - Linear models have coefficients, which can also be used to capture feature importances by considering the absolute values.

  - The supervised model that is used for feature selection doesn't need to be the same model that is used for the final supervised modeling.

scikit-learn Practice

sklearn.feature_selection.SelectFromModel

```
class sklearn.feature_selection.SelectFromModel(estimator, *, threshold=None, prefit=False, norm_order=1, max_features=None,
importance_getter='auto')                                                                                      [source]
```

Meta-transformer for selecting features based on importance weights.

New in version 0.17.

Read more in the User Guide.

Parameters:

estimator : object
The base estimator from which the transformer is built. This can be both a fitted (if `prefit` is set to True) or a non-fitted estimator. The estimator must have either a `feature_importances_` or `coef_` attribute after fitting.

threshold : string or float, default=None
The threshold value to use for feature selection. Features whose importance is greater or equal are kept while the others are discarded. If "median" (resp. "mean"), then the `threshold` value is the median (resp. the mean) of the feature importances. A scaling factor (e.g., "1.25*mean") may also be used. If None and if the estimator has a parameter penalty set to l1, either explicitly or implicitly (e.g, Lasso), the threshold used is 1e-5. Otherwise, "mean" is used by default.

prefit : bool, default=False
Whether a prefit model is expected to be passed into the constructor directly or not. If True, `transform` must be called directly and SelectFromModel cannot be used with `cross_val_score`, `GridSearchCV` and similar utilities that clone the estimator. Otherwise train the model using `fit` and then `transform` to do feature selection.

max_features : int, default=None
The maximum number of features to select. To only select based on `max_features`, set `threshold=-np.inf`.

50

scikit-learn Practice

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

```
In [38]: from sklearn.feature_selection import SelectFromModel
         from sklearn.ensemble import RandomForestClassifier

         fmodel = RandomForestClassifier()
         select = SelectFromModel(fmodel, threshold="mean")
         select.fit(X_train, y_train)

         X_train_selected = select.transform(X_train)
         X_test_selected = select.transform(X_test)
         select.get_support()
```

```
Out[38]: array([False, False,  True,  True, False, False,  True,  True, False,
                False, False, False, False, False, False, False, False,
                False, False,  True, False,  True,  True, False, False, False,
                 True, False, False])
```

```
In [39]: from sklearn.feature_selection import SelectFromModel
         from sklearn.linear_model import LogisticRegression

         fmodel = LogisticRegression()
         select = SelectFromModel(fmodel, max_features=10)
         select.fit(X_train_scaled, y_train)

         X_train_selected = select.transform(X_train_scaled)
         X_test_selected = select.transform(X_test_scaled)
         select.get_support()
```

```
Out[39]: array([False, False, False, False, False, False,  True,  True, False,
                False, False, False, False,  True, False, False, False, False,
                False,  True,  True,  True,  True,  True, False, False, False,
                 True,  True, False])
```

scikit-learn Practice

- **Example (*breast_cancer* dataset) with a prefit model**

```
In [40]:  from sklearn.ensemble import RandomForestClassifier

          fmodel = RandomForestClassifier()
          fmodel.fit(X_train, y_train)
          fmodel.feature_importances_
```

```
Out[40]:  array([0.02682935, 0.01060311, 0.08403668, 0.04123777, 0.00616597,
                 0.00596787, 0.07010536, 0.07115943, 0.00311939, 0.00456805,
                 0.01583253, 0.00443204, 0.00407996, 0.02819866, 0.00309309,
                 0.00259529, 0.00667003, 0.00183691, 0.00294645, 0.00462444,
                 0.08767875, 0.015159  , 0.08593663, 0.13601229, 0.01310815,
                 0.01984017, 0.03486298, 0.1958249 , 0.00879713, 0.00467762])
```

```
In [41]:  from sklearn.feature_selection import SelectFromModel

          select = SelectFromModel(fmodel, prefit=True, threshold="mean")
          X_train_selected = select.transform(X_train)
          X_test_selected = select.transform(X_test)
          select.get_support()
```

```
Out[41]:  array([False, False,  True,  True, False, False,  True,  True, False,
                 False, False, False, False, False, False, False, False, False,
                 False, False,  True, False,  True,  True, False, False,  True,
                  True, False, False])
```

Discussion

- If you are unsure when selecting what to use as input to your machine learning algorithms, automatic feature selection can be quite helpful.

- It is also great for reducing the amount of features needed—for example, to speed up prediction or to allow for more interpretable models.

- In most real-world cases, applying feature selection is unlikely to provide large gains in performance.

- However, it is still a valuable tool in the toolbox of the feature engineer.

Utilizing Expert Knowledge

Utilizing Expert Knowledge

- **While the purpose of machine learning in many cases is <u>to avoid having to create a set of expert-designed rules</u>, that doesn't mean that prior knowledge of the application or domain should be discarded.**

- **Feature engineering is often an important place to use *expert knowledge* for a particular application.**
  - Often, domain experts can help in identifying useful features that are much more informative than the initial representation of the data.
  - Prior knowledge about the nature of the task can be encoded in the features to aid a machine learning algorithm
  - Adding a feature does not force a machine learning algorithm to use it, and even if turns out to be noninformative, augmenting the data with this information doesn't hurt.

Utilizing Expert Knowledge

- *Example*: Time-Series Prediction

    - When evaluating a prediction task on a time series, we usually want to *learn from the past* and *predict for the future*.

    - **Our "Expert Knowledge" → New Features**

        - Two factors seem to be very important: the time of day and the day of the week.

        - The patterns for weekdays and weekends also seem to be quite different.
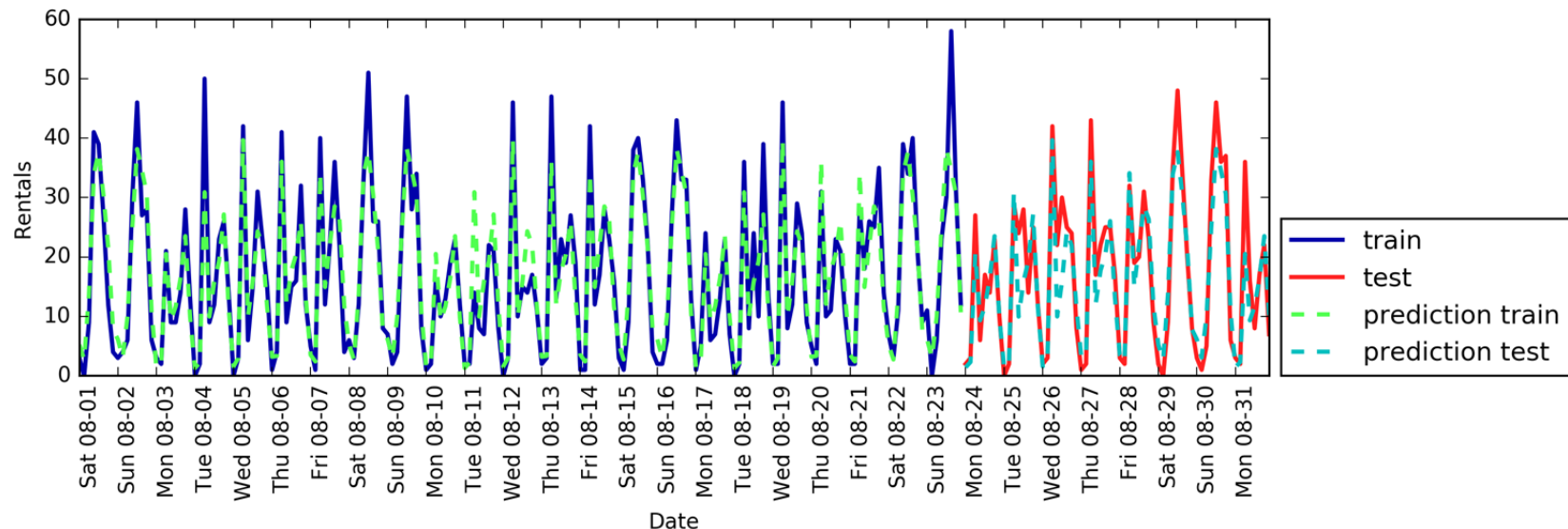
Utilizing Expert Knowledge

- *Example*: **Semiconductor Die-Level Failure Prediction**

    - The semiconductor manufacturing process consists of four basic steps: wafer fabrication, wafer testing, assembly, and final testing.

    - We want to predict the final test result at the die level before assembly using wafer test features.

    - **Our "Expert Knowledge" → New Features**

        - Three types of features related to wafer map characteristics are introduced.

        - The use of these derived features significantly improves prediction performance with statistical significance.

(1) Distance of the Die From the Wafer Center
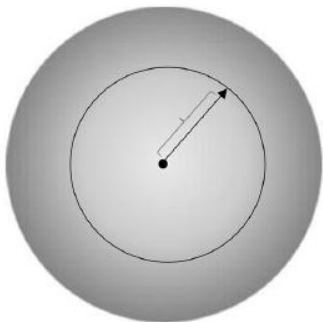
(2) Previous Final Yield at the Die Position
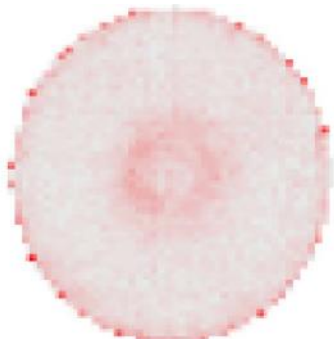
(3) Wafer Test Pass Rate of the Adjacent Dies
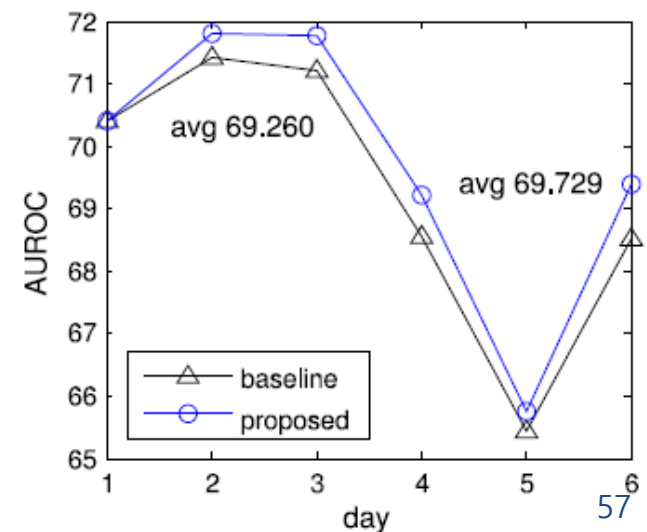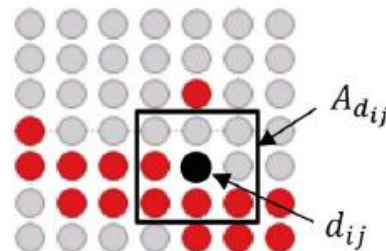
Summary and Outlook

Summary and Outlook

- **We've discussed …**

  - **How to deal with different data types** (in particular, with categorical features)

  - **The importance of representing data** in a way that is suitable for the machine learning algorithm—for example, by one-hot encoding categorical features.

  - **The importance of engineering new features**
    - linear models might benefit greatly from generating new features via binning and adding polynomials and interactions.
    - more complex, nonlinear models like random forests and SVMs might be able to learn more complex tasks without explicitly expanding the feature space.

  - **The possibility of utilizing expert knowledge** in creating derived features from your data.

- **The features that are used (and the match between features and method) are often the most important piece in making a machine learning approach work well.**

Deep Learning

Deep Learning

- A family of algorithms known as **neural networks** has recently seen a revival under the name "**deep learning**."

ARTIFICIAL INTELLIGENCE
A program that can sense, reason, act, and adapt

MACHINE LEARNING
Algorithms whose performance improve as they are exposed to more data over time

DEEP LEARNING
Subset of machine learning in which multilayered neural networks learn from vast amounts of data
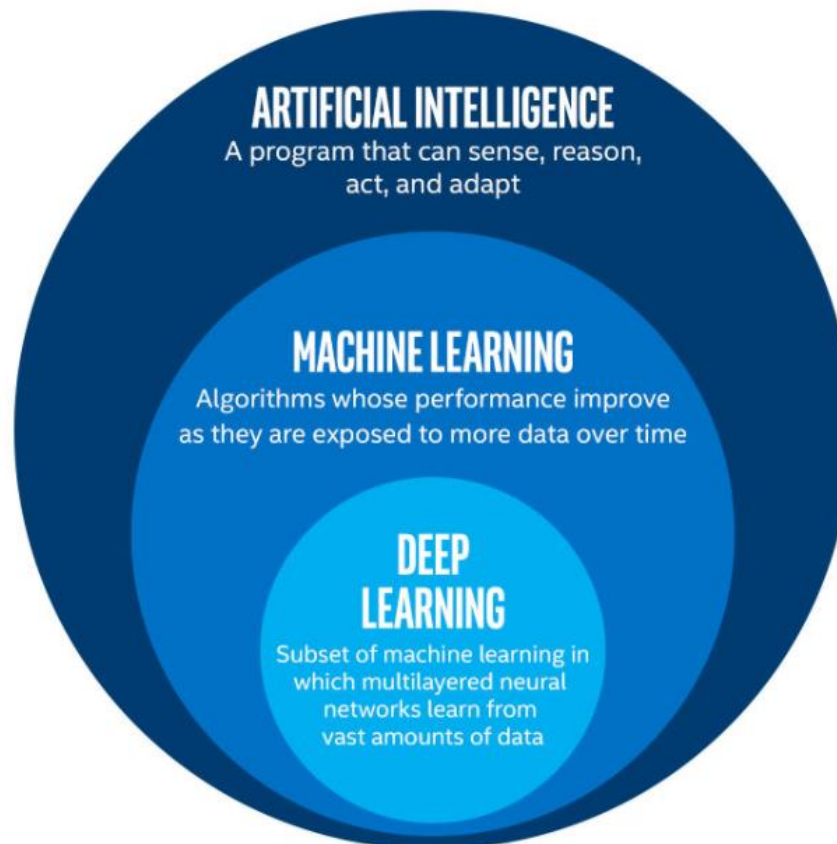
Deep Learning

- **Drawback of Traditional Machine Learning**

  - Machine learning requires domain expertise to design hand-craft feature extractor that transforms the raw data into a suitable representations (so called *feature engineering*)

    - This process is time-consuming, brittle, and not scalable in practice.

    - Handcrafted features may miss some important characteristics and complex relationships in the data.

| Input | → | Feature Extractor | → | Features | → | Traditional ML Algorithm | → | Output |

Traditional Machine Learning Flow

Performance vs. Amount of Data
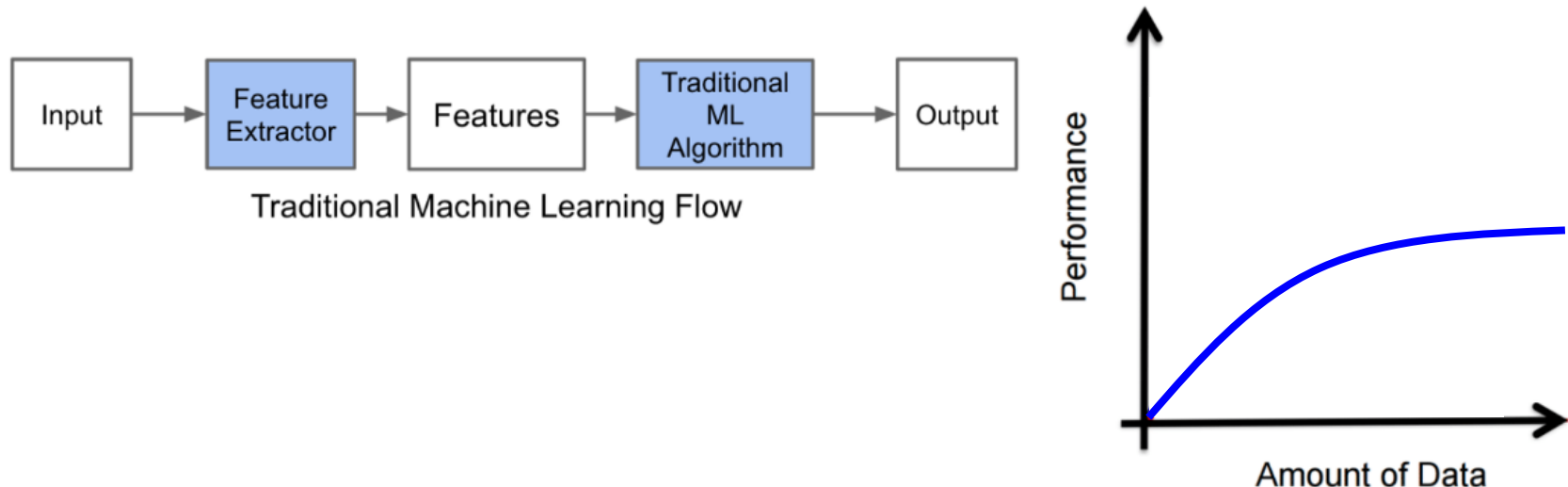
Deep Learning

- **Main Characteristics of Deep Learning**

  - **Deep Learning** is based on a cascade of multiple layers of nonlinear processing units for feature extraction and transformation.

  - Higher layers of representation amplify important aspects of the input and suppress irrelevant variations.

  - The representations are not designed by human engineers, but are learned from raw data using a general-purpose learning procedure.
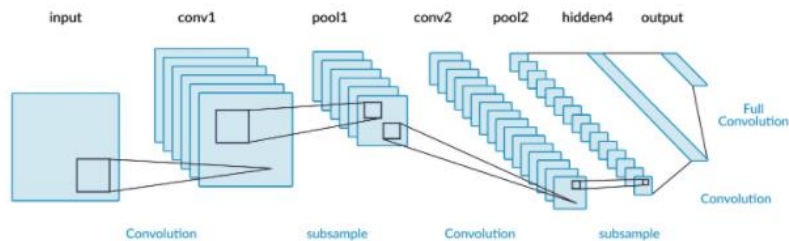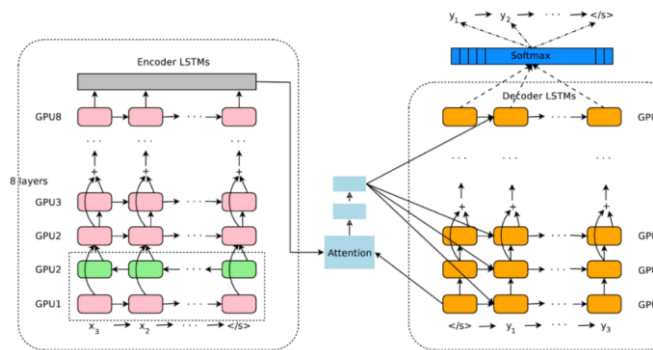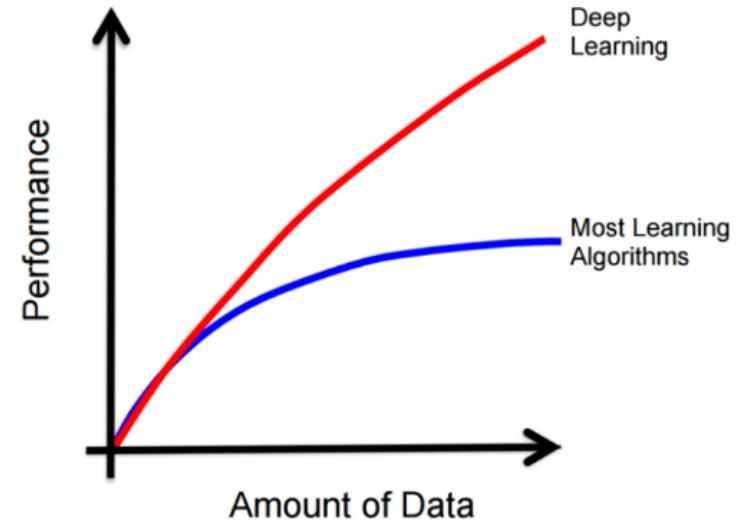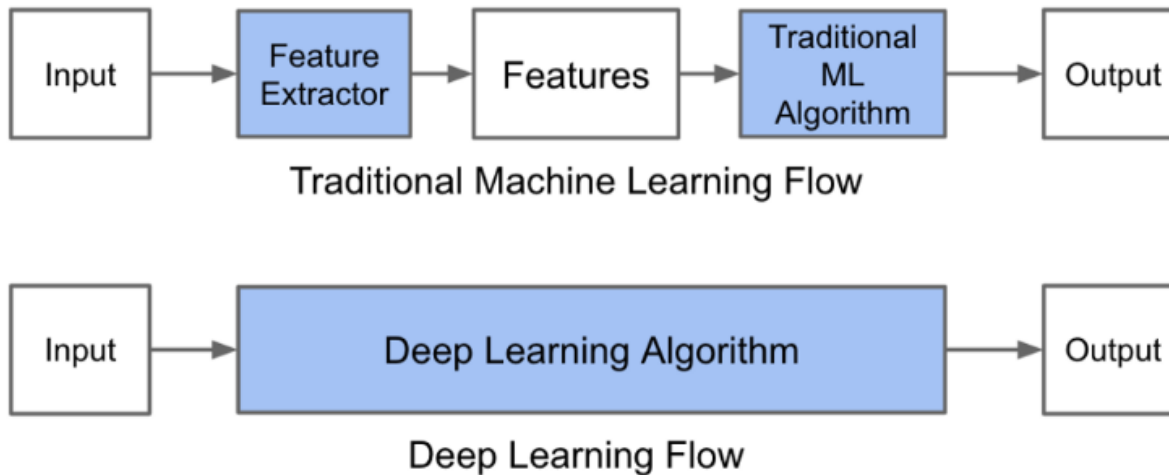

< Convolutional Neural Network >   < Recurrent Neural Network >   < Transformer >
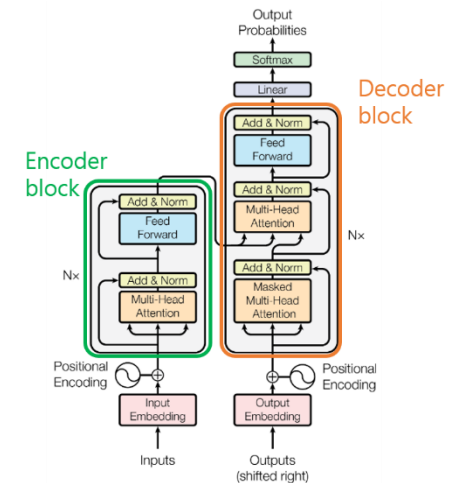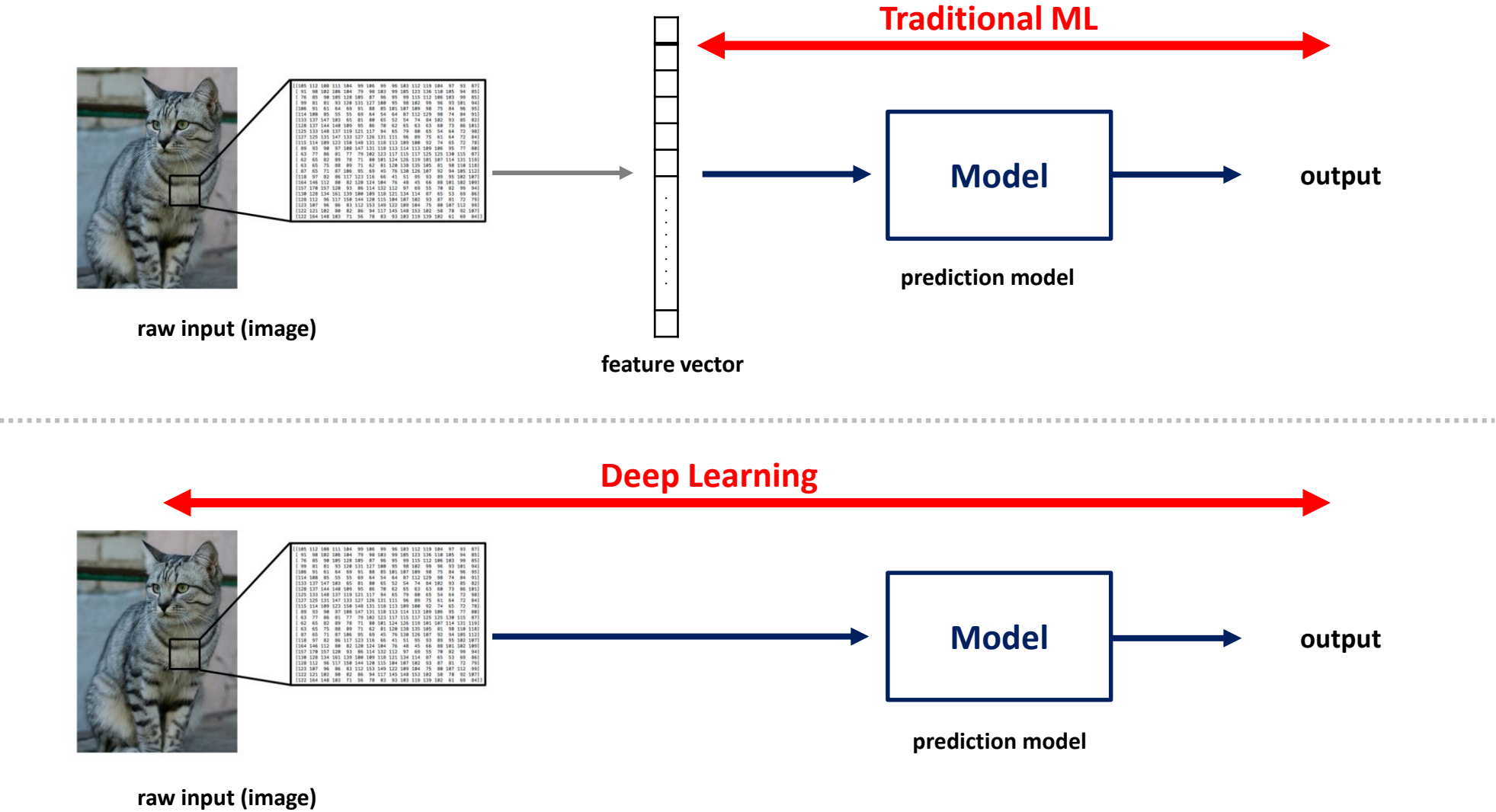
Deep Learning

- **Main Philosophy of Deep Learning**

  - **Direct Processing of Raw Input:** Deep learning algorithms are designed to process raw data directly, requiring little to no manual feature engineering.

  - **End-To-End Learning:** Seamless learning process from raw data to final outputs without the need for handcrafted intermediate steps.
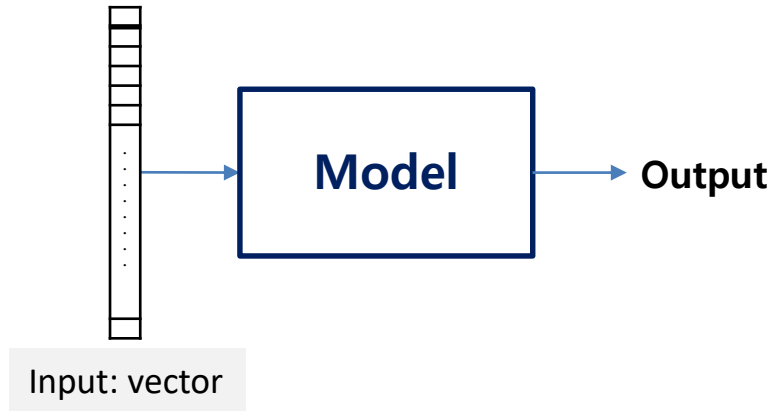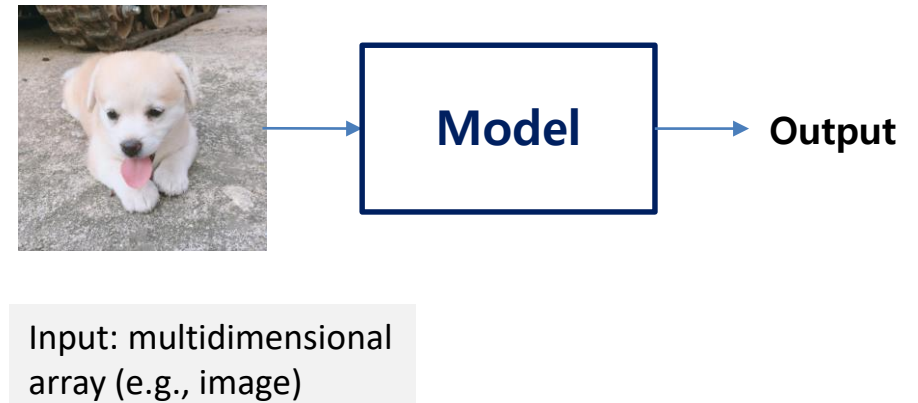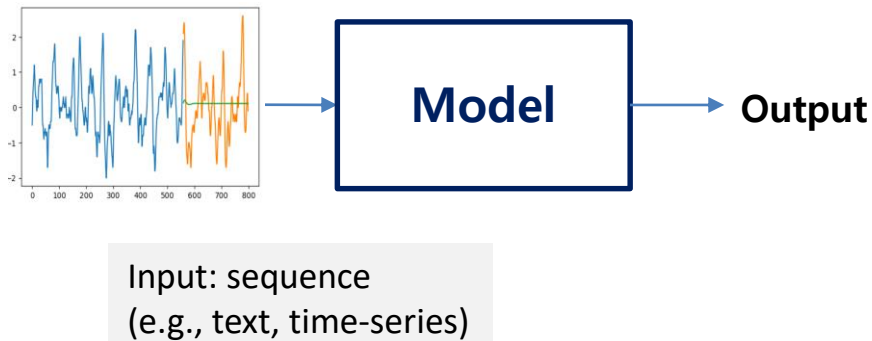
Deep Learning

raw input (image)   feature vector   **Traditional ML**   **Model**   output

prediction model

Deep Learning

raw input (image)   **Model**   output

prediction model

Deep Learning

Feed-Forward Neural Network

Input: vector

Convolutional Neural Network / Vision Transformer

Input: multidimensional array (e.g., image)

Recurrent Neural Network / Transformer

Input: sequence (e.g., text, time-series)

Graph Neural Network

Input: graph
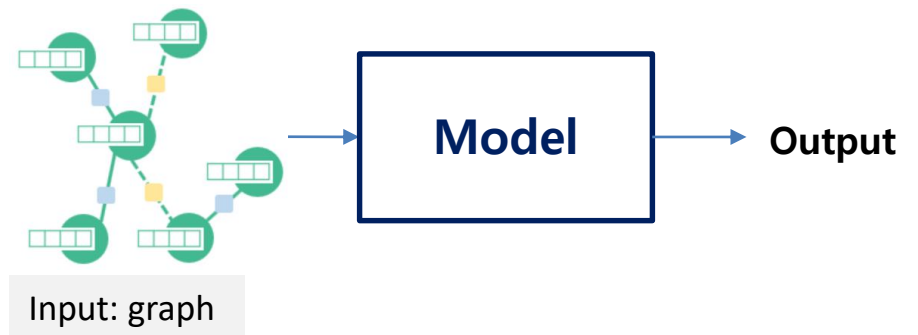
Deep Learning

Multi-Modal Neural Network