Northeastern University

DS 5110 – Lecture 9
Scikit-Learn

Roi Yehoshua

Agenda

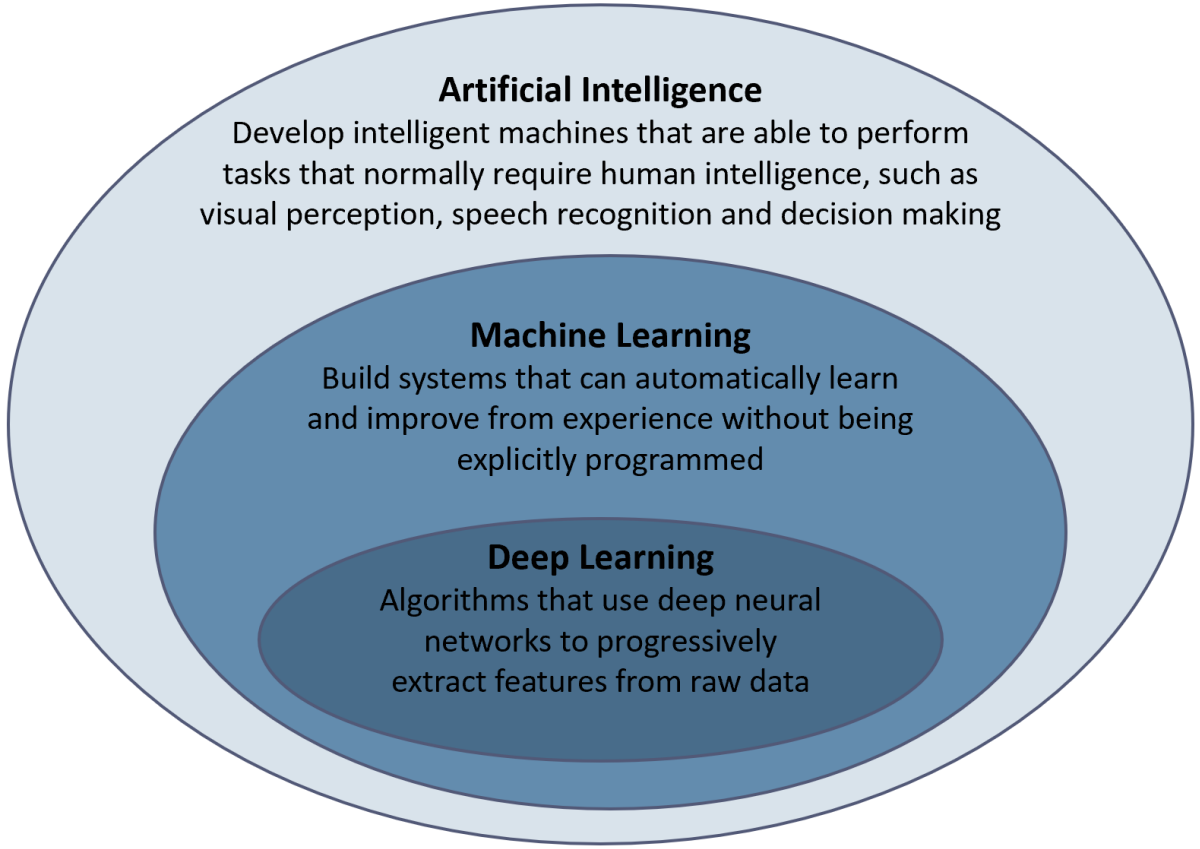
Northeastern University

- Introduction to machine learning
- The machine learning workflow
- Classification vs. regression
- Bias-variance tradeoff
- Scikit-learn
- Data preprocessing
- Model evaluation
- Pipelines
- Hyperparameter tuning
- Feature engineering

What is Machine Learning (ML)?

Northeastern University

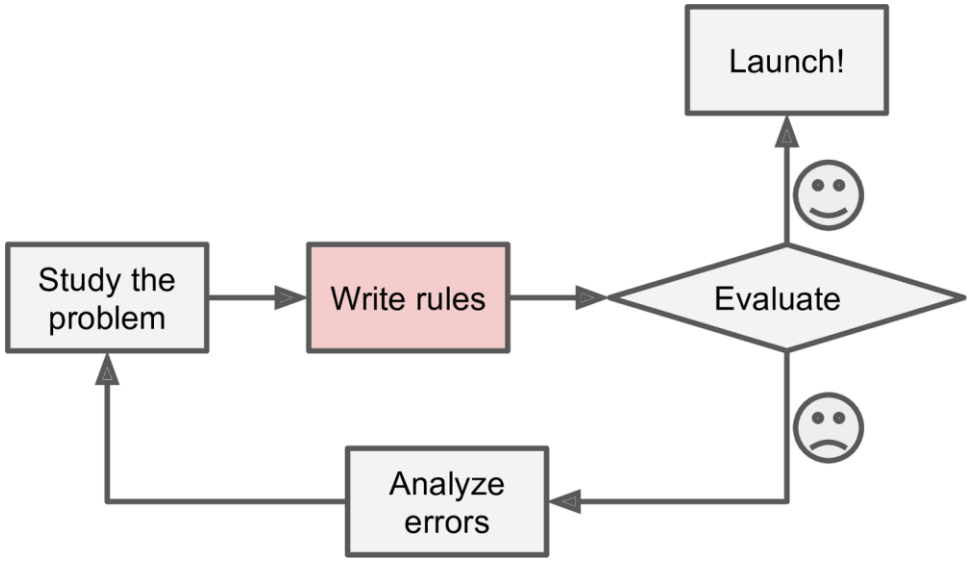
- Subfield of AI that deals with automatic learning from data



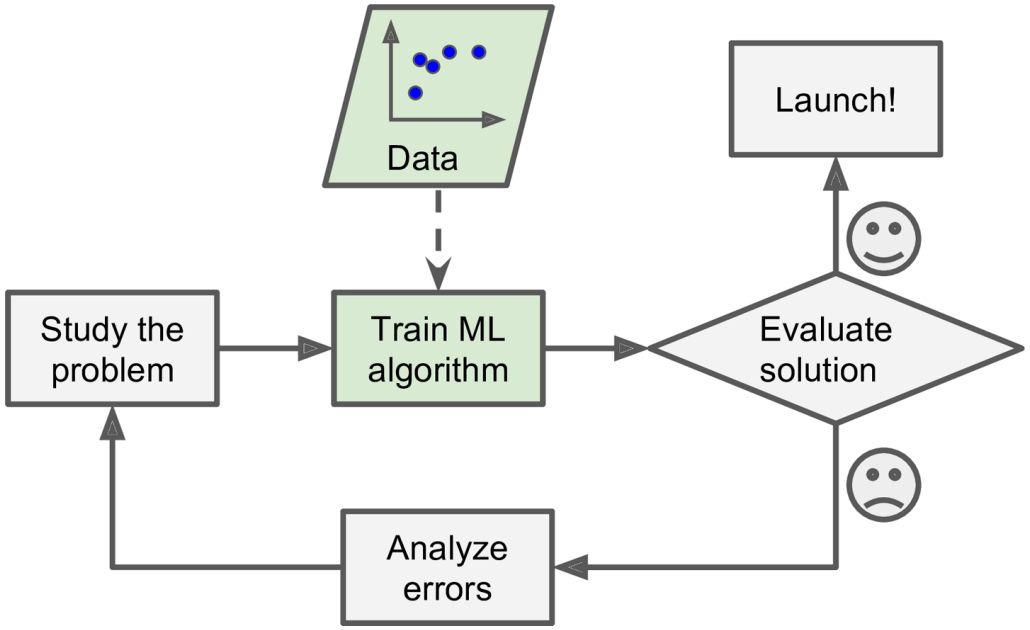
What is Machine Learning (ML)?

Northeastern University

- Solving problems with ML vs. the traditional programming tools:



Traditional approach

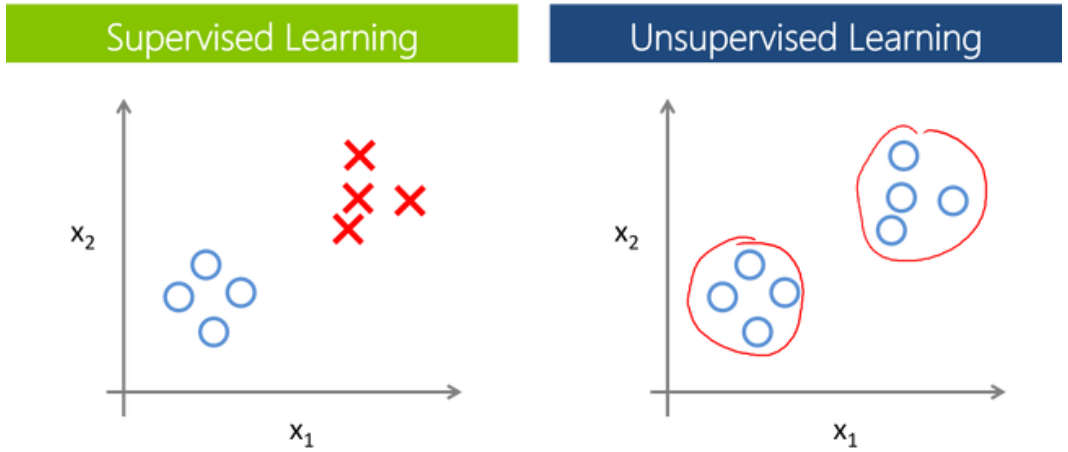


The ML approach

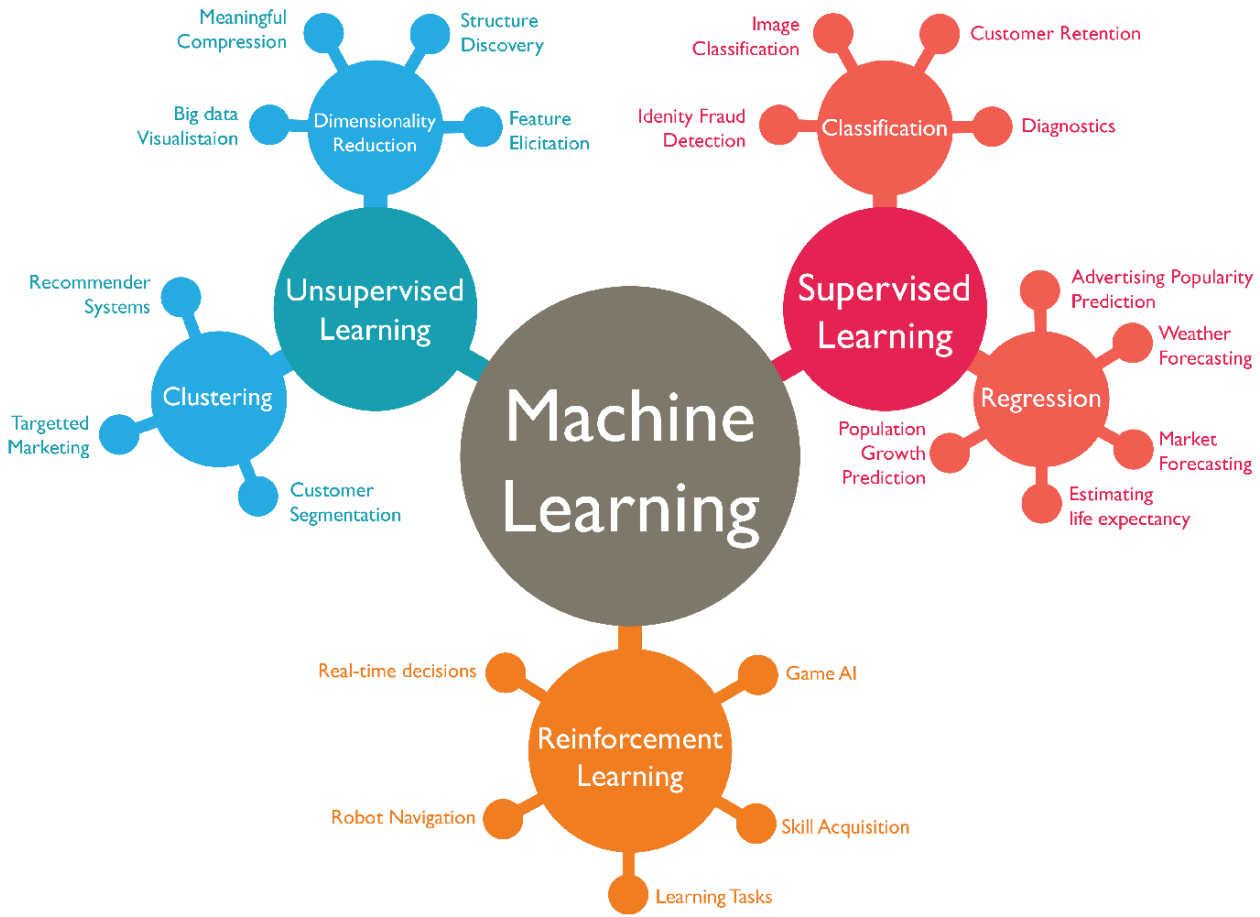
Types of Machine Learning

Northeastern University

- **Supervised learning** – learn from **labeled data**
  - Goal is to learn a function that maps an input to an output
- **Unsupervised learning** – learn from **unlabeled data**
  - Goal is to extract useful patterns from the data
- **Reinforcement learning** – learn to take the best actions in an environment in order to maximize a cumulative reward



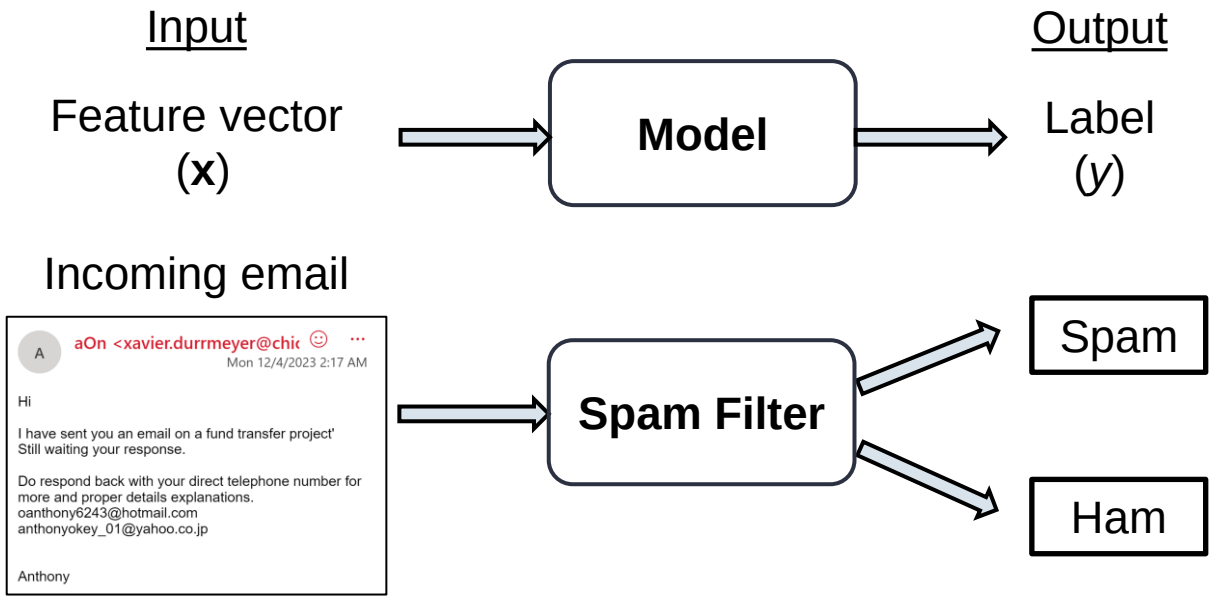
Types of Machine Learning



Supervised Learning

Northeastern University

- **Given**: a **training set** of n labeled examples $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$
  - Each $\mathbf{x}_i$ is a d-dimensional vector of feature values, $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{id})^T$
  - y_i is the **target** or **label** we are trying to predict
- **Goal**: learn a function that maps **x** to y
- The trained model is evaluated on an **independent test set**



Features Matrix

- The dataset is typically stored in a single matrix called features / design matrix
- Each row of the matrix is one data point (one feature vector)
- Each column represents the values of a given feature across all the data points

Feature 1 Feature *d*

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{i1} & x_{i2} & \cdots & x_{id} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{pmatrix}$$

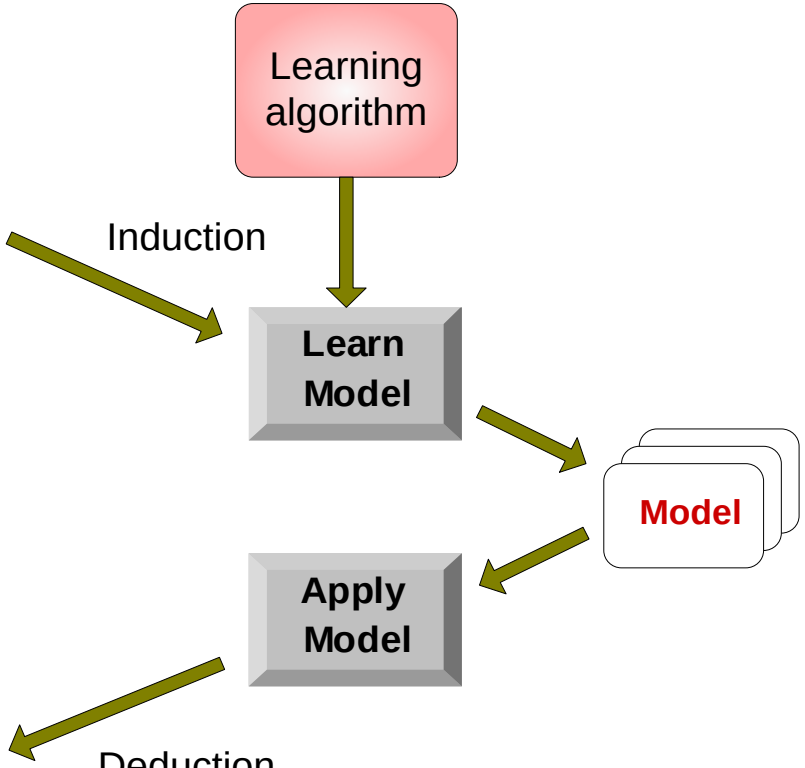
Training example *i*

Supervised Learning Workflow

Northeastern University

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

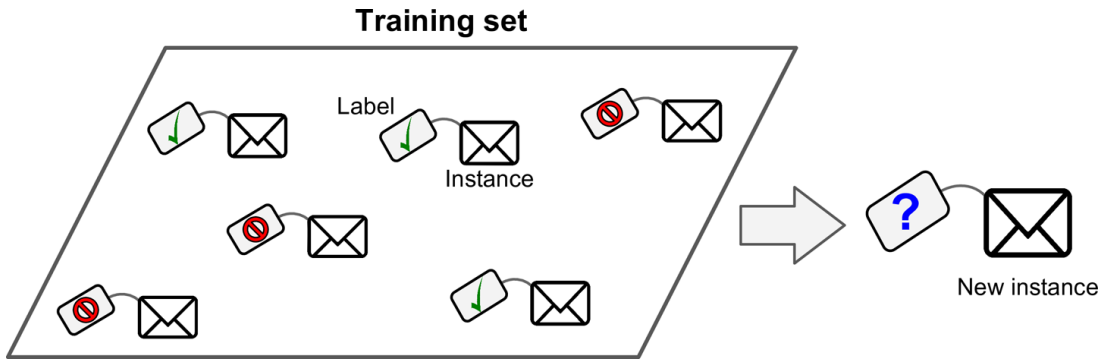


Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set

Classification vs. Regression

- When the label *y* is a discrete variable, the learning problem is called **classification**
  - *y* is the **class** that the sample belongs to
- When the label *y* is continuous, the learning problem is called **regression**



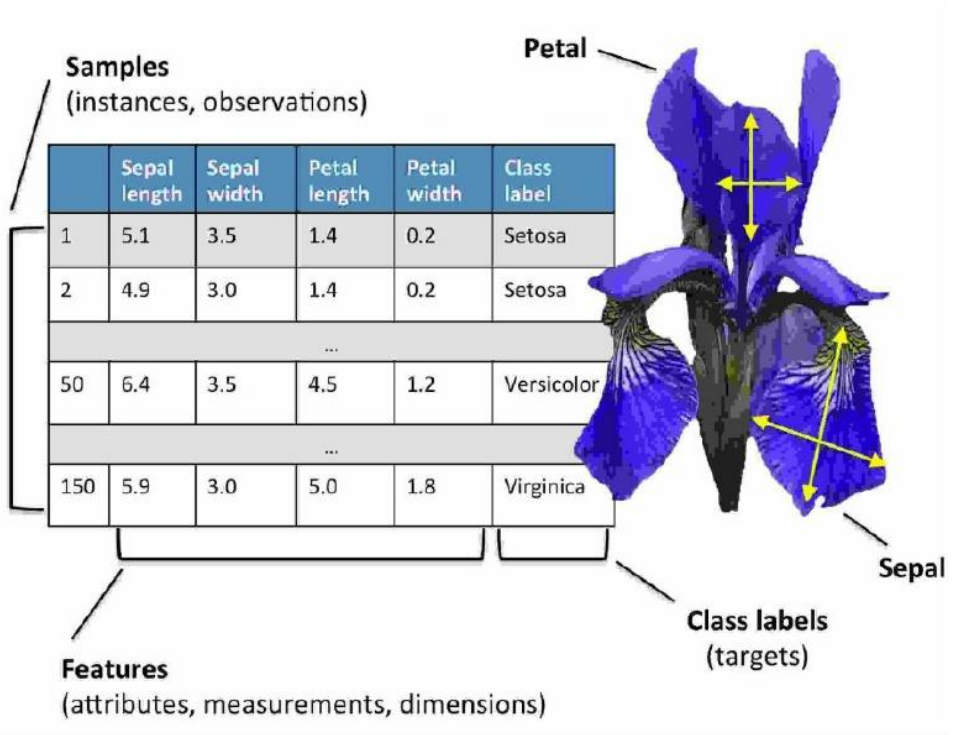
Classification

Regression

Example 1: Iris Flower Classification

Northeastern University

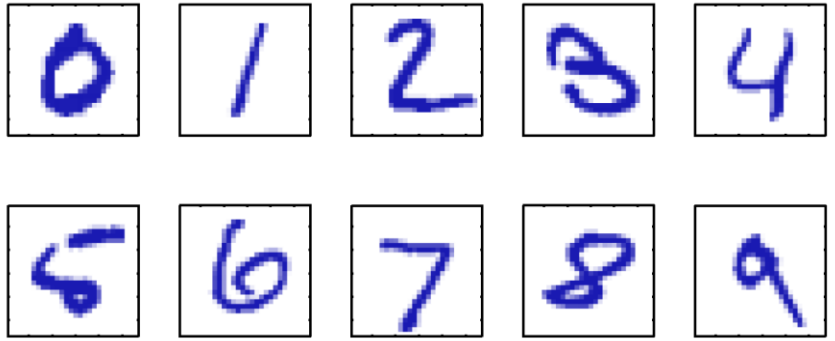
- From the statistician Douglas Fisher (1936)
- Data set: 150 Iris flowers, 50 from each of the species: Setosa, Virginica, Versicolour
- Four attributes
  - Sepal width and length (in centimeters)
  - Petal width and length (in centimeters)



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

Example 2: Handwritten Digit Recognition

Northeastern University



- MNIST data set
- **Input**: images of 28 × 28 pixels
  - Pixel values range from 0 to 255 (grey level)
- **Output**: a digit 0-9
- Data set contains 60,000 training examples and 10,000 test examples
- **Setup**:
  - Represent each input image as a vector $\mathbf{x} \in R^{784}$
  - Learn a classifier $h(\mathbf{x})$ such that h: $\mathbf{x} \rightarrow \{0,1,2,3,4,5,6,7,8,9\}$
- One of the first commercial and widely used ML systems (for zip codes and checks)

Example 3: Credit Approval

Northeastern University

- Input: applicant information

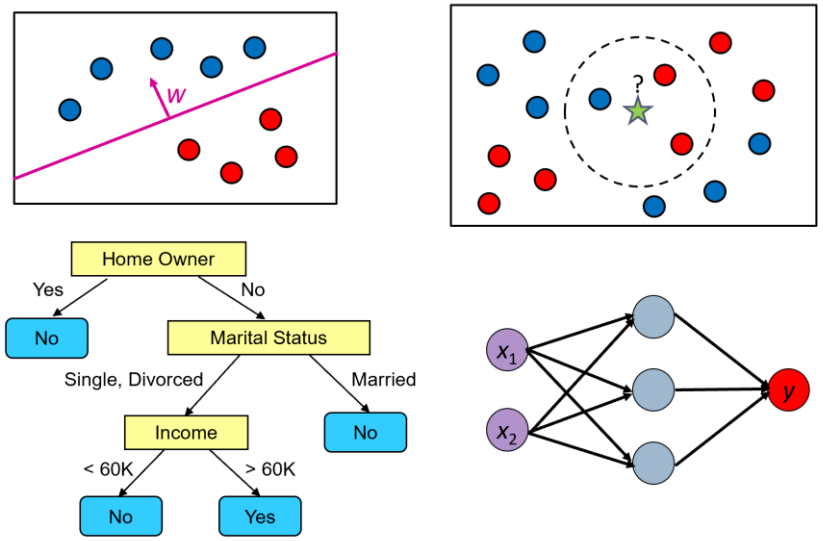
Relation: german_credit

No.	checking_status Nominal	duration Numeric	credit_history Nominal	purpose Nominal	credit_amount Numeric	savings_status Nominal	employment Nominal	installment_commitment Numeric	personal_status Nominal
1	(0	6.0	critical/other exi...	radio/tv	1169.0	no known savi...	)=7	4.0	male single
2	0(=X(200	48.0	existing paid	radio/tv	5951.0	(100	1(=X(4	2.0	female div/dep...
3	no checking	12.0	critical/other exi...	education	2096.0	(100	4(=X(7	2.0	male single
4	(0	42.0	existing paid	furnitu...	7882.0	(100	4(=X(7	2.0	male single
5	(0	24.0	delayed previously	new car	4870.0	(100	1(=X(4	3.0	male single
6	no checking	36.0	existing paid	education	9055.0	no known savi...	1(=X(4	2.0	male single

- Output:
  - approve credit? → classification
  - credit line (dollar amount) → regression

Learning Algorithms

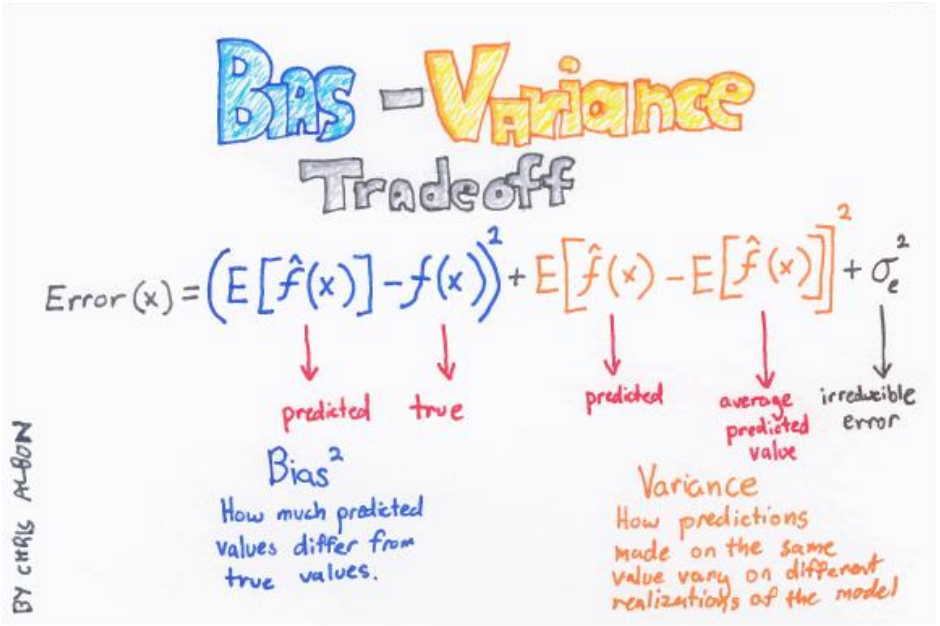
- There are many supervised learning algorithms
  - From simple decision trees to deep neural networks

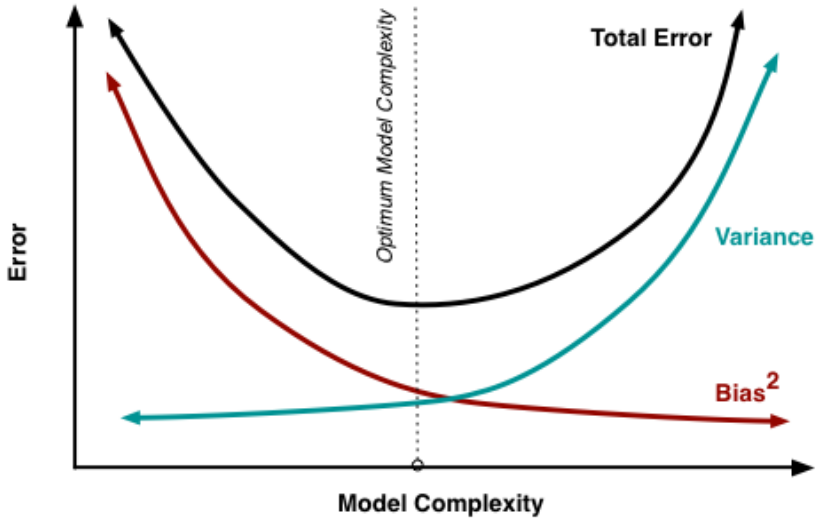


- **No free lunch (NFL)** theorem:
  - No single learning algorithm performs best across all possible problems or datasets
- **Occam's Razor**:
  - Given multiple explanations (models) for a phenomenon, prefer the simplest one

Bias-Variance Tradeoff

- The generalization error of a model can be decomposed into three parts
  - **Bias** measures the error of the model on the training data
  - **Variance** captures the sensitivity of the model to fluctuations in the training data
  - **Irreducible error** represents the noise inherent in the data

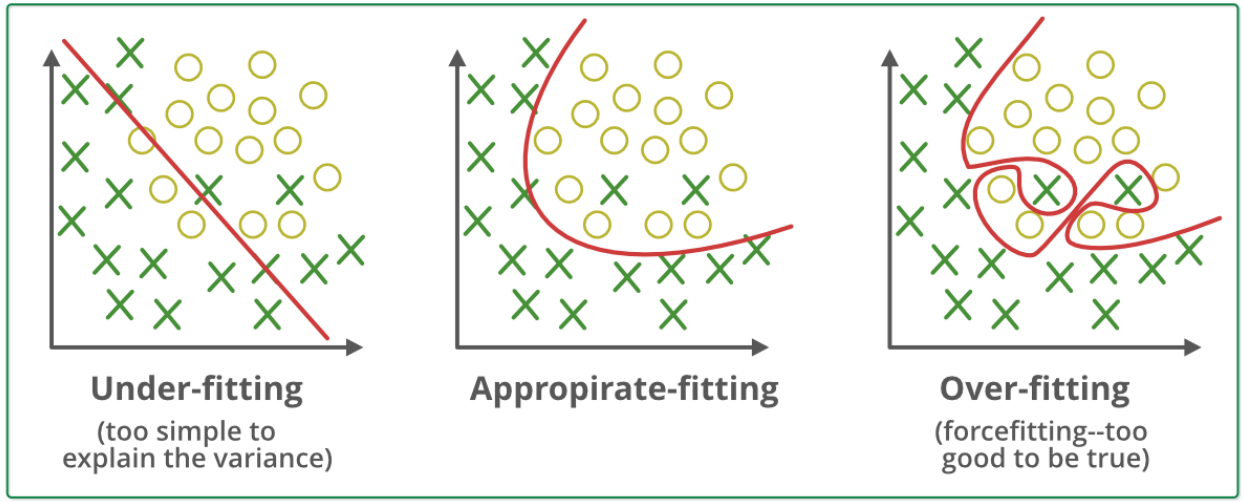




Overfitting and Underfitting

Northeastern University

- **Underfitting**: the model is too simple to capture the pattern in the data
  - High bias, low variance
- **Overfitting**: the model learns the training data too well and fails to generalize
  - Low bias, high variance
- It's challenging to find the right balance between bias and variance



Scikit-Learn

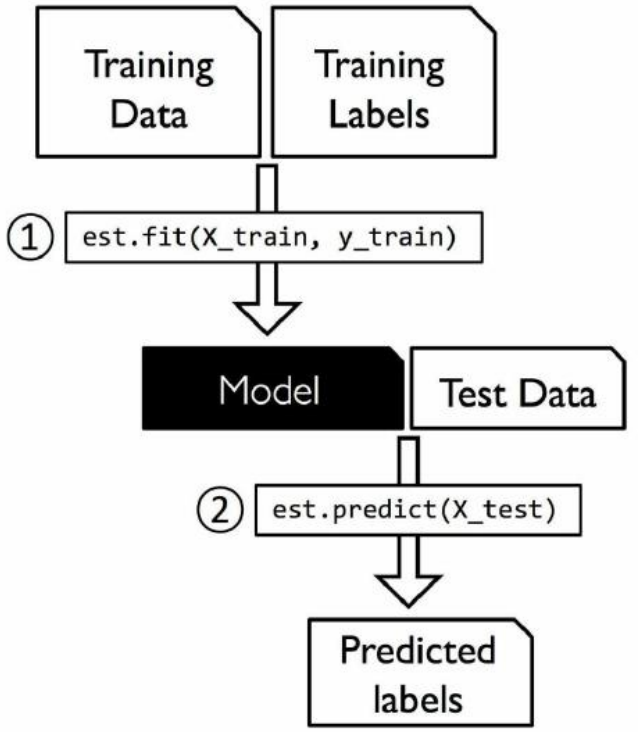
Northeastern University

- scikit-learn is a free machine learning library for Python
- Provides efficient versions of a large number of common ML algorithms, including various classification, regression and clustering algorithms
- Characterized by a clean, uniform, and streamlined API, as well as by very useful and complete online documentation
- Designed to interoperate with Python scientific libraries NumPy and SciPy
- Largely written in Python, with some core algorithms written in Cython to achieve fast performance
- http://scikit-learn.org/

Scikit-Learn Estimator API

Northeastern University

- An **estimator** is any object that learns from data
- All estimators have a **fit()** method that learns the model's parameters from the data
- Two main types of estimators:
  - **Predictors** can also make predictions on new data
    - Also have **predict()** and **score()** methods
  - **Transformers** make changes to the data (e.g., normalization)
    - Also have a **transform()** method



Iris Classification Example

Northeastern University

- First step is to load the dataset
- Typically the dataset is loaded from external file (e.g., csv) via Pandas
- Scikit-Learn also provides some standard datasets in **sklearn.datasets**

load_boston(*[, return_X_y])	Load and return the boston house-prices dataset (regression).
load_iris(* [, return_X_y, as_frame])	Load and return the iris dataset (classification).
load_diabetes(* [, return_X_y, as_frame])	Load and return the diabetes dataset (regression).
load_digits(* [, n_class, return_X_y, as_frame])	Load and return the digits dataset (classification).
load_linnerud(* [, return_X_y, as_frame])	Load and return the physical excercise linnerud dataset.
load_wine(* [, return_X_y, as_frame])	Load and return the wine dataset (classification).
load_breast_cancer(* [, return_X_y, as_frame])	Load and return the breast cancer wisconsin dataset (classification).

Loading the Dataset

Northeastern University

- Let’s load the Iris dataset from Scikit-Learn:

```
from sklearn.datasets import load_iris

X, y = load_iris(as_frame=True, return_X_y=True)
X.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Data Exploration

Northeastern University

- Understand the data types of the features
  - e.g., numerical, categorical
- Understand statistical properties of the features
- Examine correlations among the features
- Examine correlations between the features and the target
- Identify potential issues, such as missing values, outliers, imbalanced classes

Data Exploration

Northeastern University

- The **info()** method lets you inspect the data types and check for missing values

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
dtypes: float64(4)
memory usage: 4.8 KB
```

Data Exploration

Northeastern University

- The **describe()** method provides summary statistics for each feature

```
X.describe()
```

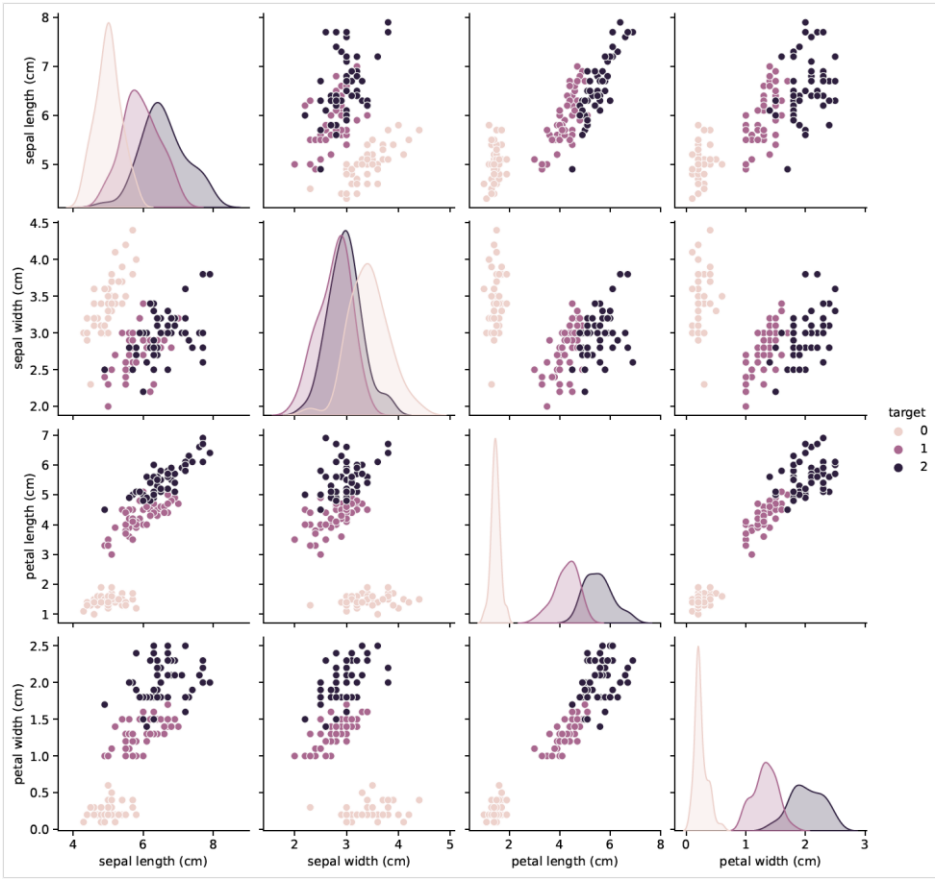
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Data Exploration

Northeastern University

- SeaBorn's **pairplot()** visualizes the feature distributions and their relationships

```
df = pd.concat([X, y], axis=1)
sns.pairplot(df, hue='target');
```

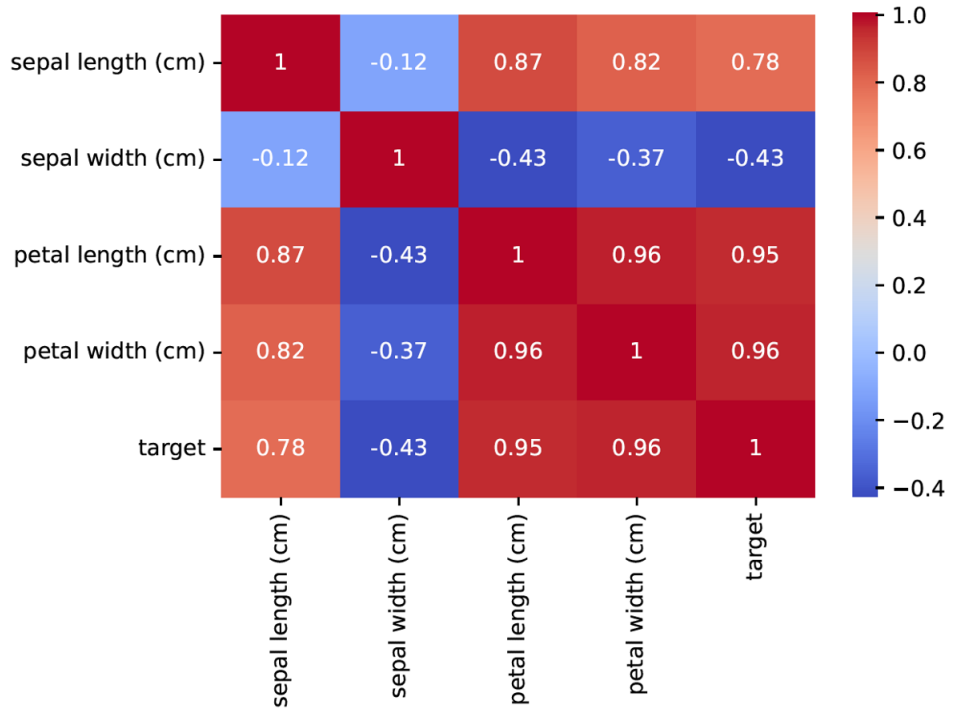


Data Exploration

Northeastern University

- The **corr()** method computes correlation coefficients between the features:

```
sns.heatmap(df.corr(), annot=True, cmap='coolwarm');
```



Data Exploration

Northeastern University

- Check class imbalance:

```
y.value_counts()
```

[11]:

```
target
0    50
1    50
2    50
Name: count, dtype: int64
```

Train-Test Split

Northeastern University

- The test set is used to evaluate the model's performance on unseen data
  - It should remain untouched until the final model evaluation
- You can use Scikit-Learn's **train_test_split()** for splitting the data to train/test:
  - The default split is 75% train, 25% test
  - You can change this using the **train_size** or **test_size** parameters
  - Set **stratify=y** to maintain the same class proportions in the test set

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, stratify=y)
```

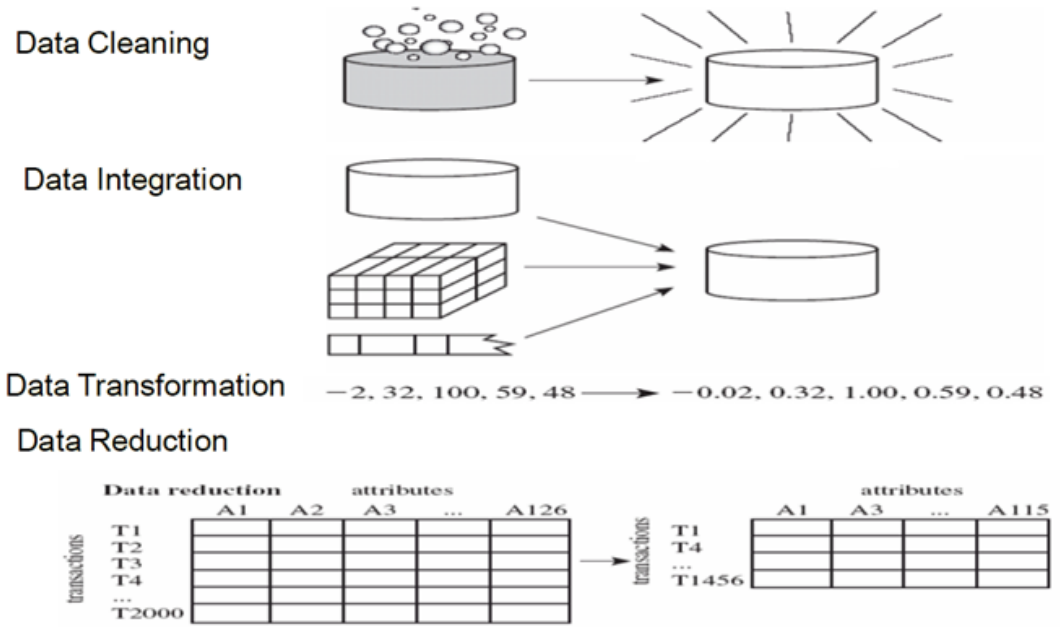
```
print('X_train shape:', X_train.shape)
print('X_test shape:', X_test.shape)
```

```
X_train shape: (112, 4)
X_test shape: (38, 4)
```

Data Preprocessing

Northeastern University

- Data cleaning
- Handling missing data
- Encoding categorical data
- Discretization
- Scaling / normalization
- Outlier detection
- Dimensionality reduction

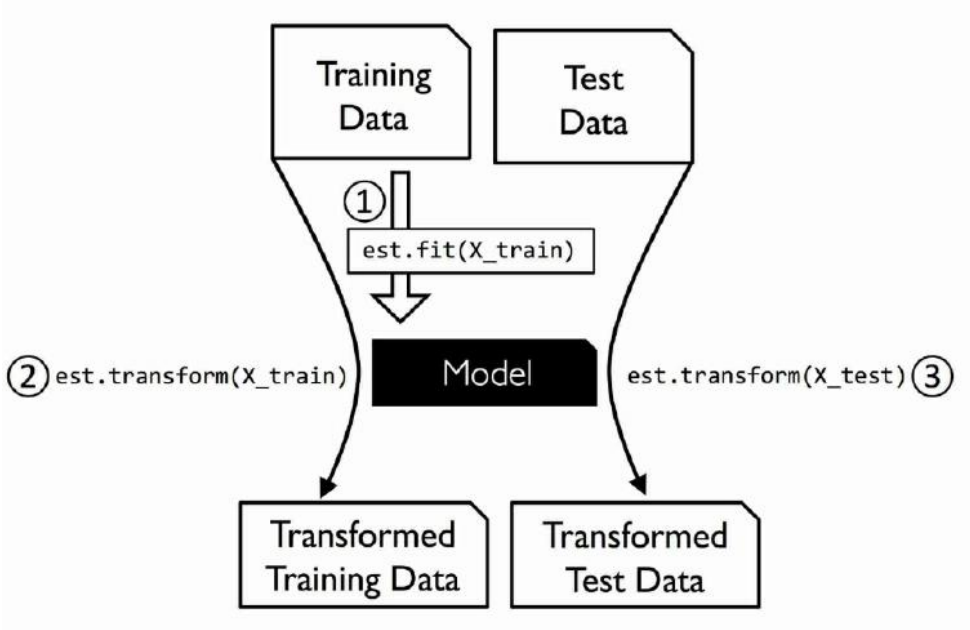


- **sklearn.preprocessing** includes many methods for data preprocessing

Scikit-Learn Transformers

Northeastern University

- Transformers are objects used to preprocess and transform the data
- Transformers provide the following methods:
  - **fit()** - learns model parameters from the data set
    - e.g., mean and std for feature scaling
  - **transform()** - applies the transformation to the data
  - **fit_transform()** - calls both fit() and transform()



Example: Feature Scaling

Northeastern University

- Standardization transforms each feature to have zero mean and unit variance

$$x' = \frac{x - \overline{x}}{\sigma}$$

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

```
X_train_scaled[:5]
```

```
array([[ 1.79213839, -0.60238047,  1.31532306,  0.92095427],
       [ 2.14531053, -0.60238047,  1.65320421,  1.05135487],
       [-0.4446185 , -1.50797259, -0.03620155, -0.25265117],
       [ 0.26172578, -0.60238047,  0.13273902,  0.13855064],
       [-0.4446185 , -1.28157456,  0.13273902,  0.13855064]])
```

Training the Model

Northeastern University

- Create an instance from the estimator you want to use
  - e.g., LogisticRegression, RandomForestClassifier
- Set its **hyperparameters** in the constructor
  - Hyperparameters are configuration settings set before training a model
  - In contrast to parameters that are learned from the data during training
- Call its **fit()** method on the training set

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(random_state=42)
clf.fit(X_train_scaled, y_train)
```

Training the Model

Northeastern University

- You can find a list of all the model's hyperparameters on its Scikit-Learn page
  - e.g., https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

```
class sklearn.tree.DecisionTreeClassifier(*, criterion='gini', splitter='best',
max_depth=None, min_samples_split=2, min_samples_leaf=1,
min_weight_fraction_leaf=0.0, max_features=None, random_state=None,
max_leaf_nodes=None, min_impurity_decrease=0.0, class_weight=None, ccp_alpha=0.0,
monotonic_cst=None)                                                    [source]
```

A decision tree classifier.

Read more in the User Guide.

Parameters:

criterion : *{"gini", "entropy", "log_loss"}, default="gini"*

The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "log_loss" and "entropy" both for the Shannon information gain, see Mathematical formulation.

Model Inspection

Northeastern University

- After training, you can inspect the model's learned parameters
- Each estimator provide attributes that end in underscore to access these parameters
- For example, logistic regression learns a set of weights (coefficients) for each feature:

```
clf.coef_
```

```
array([[-1.049416  ,  1.1534895 , -1.62864463, -1.56781588],
       [ 0.49537647, -0.53075959, -0.26388111, -0.80838681],
       [ 0.55403953, -0.62272991,  1.89252574,  2.37620269]])
```

Model Evaluation

Northeastern University

- We can first evaluate the model's performance on the training set
- The **score()** method of the estimator returns the default scoring metric
  - In classification it is the **accuracy** (proportion of correctly classified examples)
  - In regression, it is **R^2 score** (discussed later)

```
# Evaluate the model on the training set
train_accuracy = clf.score(X_train_scaled, y_train)
print(f'Train accuracy: {train_accuracy:.4f}')
```

```
Train accuracy: 0.9643
```

Model Evaluation

Northeastern University

- To evaluate the performance on the test set we first need to preprocess it
  - Using the same transformers that were applied to the training set
  - Calling their **transform()** method instead of fit_transform()

```
# Transform the test set
X_test_scaled = scaler.transform(X_test)

# Evalute the model on the test set
test_accuracy = clf.score(X_test_scaled, y_test)
print(f'Test accuracy: {test_accuracy:.4f}')
```

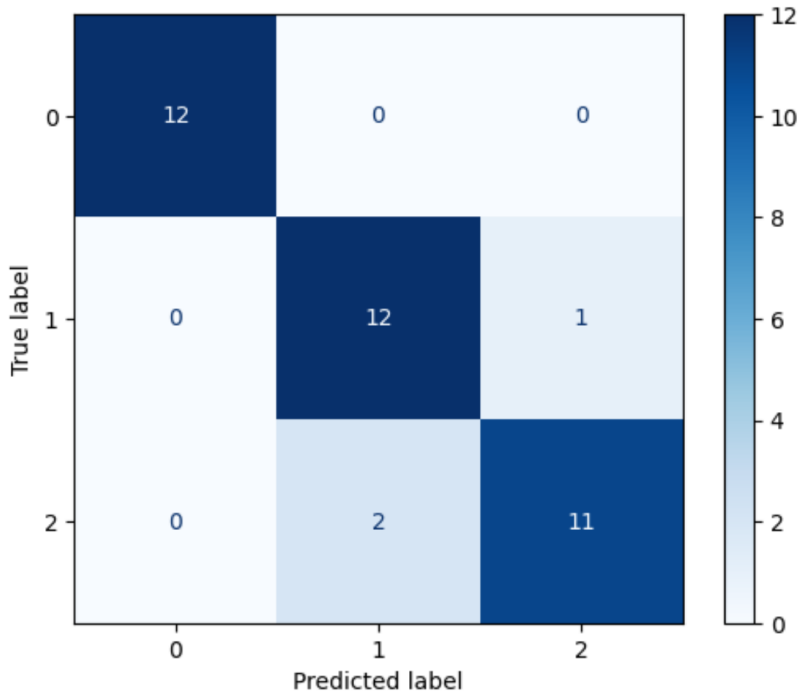
```
Test accuracy: 0.9211
```

Confusion Matrix

Northeastern University

- A useful tool to inspect the classification errors of the model

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

y_test_pred = clf.predict(X_test_scaled)
conf_mat = confusion_matrix(y_test, y_test_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=conf_mat)
disp.plot(cmap='Blues');
```



Using the Model for Predictions

Northeastern University

- We can use the estimator's **predict()** method to make predictions on new samples

```
test_sample = [X_test_scaled[5]]
y_pred = clf.predict(test_sample)
print('Predicted label:', y_pred)

true_label = y_test.iloc[5]
print('True label:', true_label)
```

```
Predicted label: [1]
True label: 1
```

Model Selection

Northeastern University

- It is important to evaluate multiple models on your dataset
  - Each model has unique strengths and weaknesses
  - It is hard to know in advance which model will perform the best
- Typically, you start from a baseline model that is fast to train
  - e.g., logistic regression, decision tree
- Then you can experiment with more complex models to see if you get better results

Model Selection

Northeastern University

- Example for comparing different classifiers on Iris:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier

# Create a dictionary of classifiers
classifiers = {
    'Logistic Regression': LogisticRegression(random_state=42),
    'KNN': KNeighborsClassifier(),
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42),
    'Gradient Boosting': GradientBoostingClassifier(random_state=42),
    'SVM': SVC(),
    'MLP': MLPClassifier(random_state=42)
}
```

Model Selection

Northeastern University

- Training all the estimators and save their results in a dictionary:

```
# Train all the estimators and save the accuracy results
results = {}

for name, clf in classifiers.items():
    clf.fit(X_train_scaled, y_train)
    results[name] = {}
    results[name]['Training Accuracy'] = clf.score(X_train_scaled, y_train)
    results[name]['Test Accuracy'] = clf.score(X_test_scaled, y_test)
```

Model Selection

Northeastern University

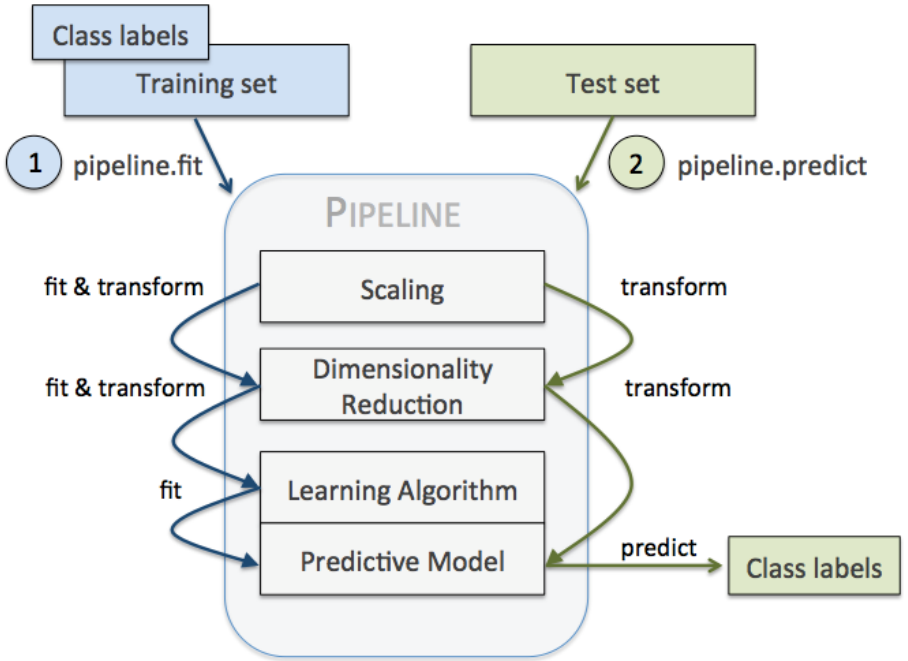
- Display the results in a DataFrame:

```
# Display the evaluation results in a DataFrame
results_df = pd.DataFrame.from_dict(results).T
results_df
```

	Training Accuracy	Test Accuracy
Logistic Regression	0.964286	0.921053
KNN	0.982143	0.921053
Decision Tree	1.000000	0.894737
Random Forest	1.000000	0.921053
Gradient Boosting	1.000000	0.973684
SVM	0.964286	0.947368
MLP	0.964286	0.868421

Pipelines

- Pipelines allow you to chain multiple transformers with a final estimator
- The pipeline automates the appropriate method calls during training/test
- Allow seamless integration with other Scikit-Learn tools for model evaluation



Pipelines

Northeastern University

- Example for defining and fitting a pipeline:

```
from sklearn.pipeline import Pipeline
model = Pipeline([
    ('scaler', StandardScaler()),
    ('clf', DecisionTreeClassifier())
])

model.fit(X_train, y_train)

train_accuracy = model.score(X_train, y_train)
print(f'Train accuracy: {train_accuracy:.4f}')
test_accuracy = model.score(X_test, y_test)
print(f'Test accuracy: {test_accuracy:.4f}')
```

```
Train accuracy: 1.0000
Test accuracy: 0.8947
```

Pipelines

Northeastern University

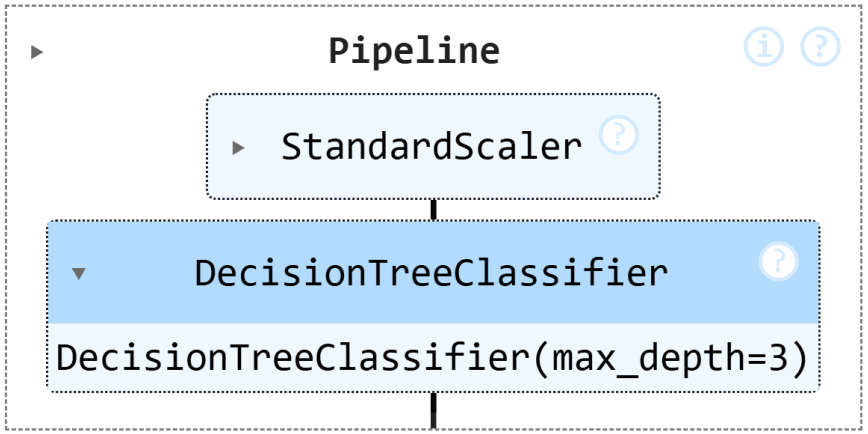
- To access the estimators inside a pipeline use its **named_steps** attribute:

```
print(model.named_steps['clf'])
```

```
DecisionTreeClassifier()
```

- To modify the hyperparameters of an estimator use the **set_params()** method:
  - The syntax for writing the parameters is <estimator name>__<parameter>

```
model.set_params(clf__max_depth=3)
```



Model Selection and Tuning

Northeastern University

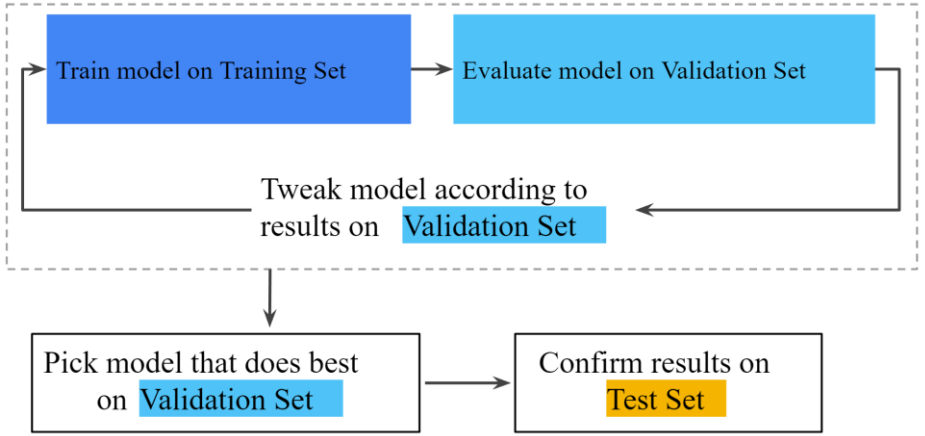
- **Goal:** choose the model that performs best on unseen data (low generalization error)
- **Model selection** takes into account both:
  - Model type (e.g., logistic regression, SVM, KNN, etc.)
  - Different model configurations (hyperparameters)
- **Model tuning** is an iterative process of tweaking the model hyperparameters and data preprocessing steps to extract the best performance out of any given model
- Using the test set for model selection or tuning can cause **overfitting to the test data**
  - Leading to overly optimistic estimates of the model performance on unseen data

Validation Set

Northeastern University

- Split the dataset into three disjoint subsets:

- Use the validation set to evaluate and compare different model configurations
- Reserve the test set for a final evaluation of the model with the best validation score

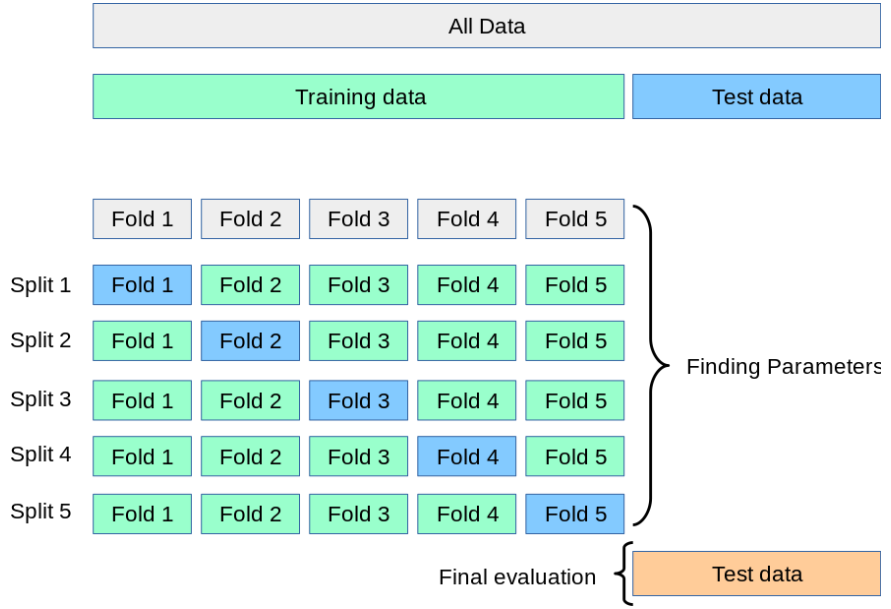


- **Problem**: we lose a decent amount of the training data for validation

k-Fold Cross Validation

Northeastern University

- Randomly split the training set into *k* equal-sized partitions S_1, …, S_k
- For $i = 1, \ldots, k$:
  - Train the model on all the training data except for S_i
  - Test the model on S_i
- The resulting *k* errors are averaged to obtain an estimate of the generalization error



Cross Validation in Scikit-Learn

Northeastern University

- The method **cross_val_score()** can be used to perform cross-validation
  - The **cv** argument specifies the splitting strategy (defaults to 5 folds)

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(model, X_train, y_train, cv=5)

print('CV scores:', np.round(scores, 4))
print(f'Average score: {scores.mean():.4f} ± {scores.std():.4f}')
```

```
CV scores: [0.913  1.     0.9091 1.     0.9091]
Average score: 0.9462 ± 0.0439
```

Hyperparameter Tuning

Northeastern University

- Search for the optimal combination of hyperparameter values
- Main methods
  - **Manual tuning**: Choose hyperparameters based on intuition, experience, or educated guess
  - **Grid search**: Perform an exhaustive search over all possible hyperparameter combinations
  - **Random search**: Randomly sample combinations of hyperparameter values
  - **Bayesian optimization**: Use probabilistic models to guide the search for better hyperparameters by balancing exploration and exploitation

Grid Search In Scikit-Learn

Northeastern University

- Can be performed using the **GridSearchCV** class
  - Requires defining a grid of parameters
  - Uses cross-validation to evaluate each combination of parameters
- For example, we can use grid search to find optimal parameters of the decision tree:

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': np.arange(1, 11),
    'min_samples_leaf': np.arange(1, 11)
}

clf = DecisionTreeClassifier(random_state=42)
grid_search = GridSearchCV(clf, param_grid, cv=3, n_jobs=-1)

grid_search.fit(X_train, y_train)
print(grid_search.best_params_)
```

```
{'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf': 1}
```

Grid Search In Scikit-Learn

Northeastern University

- You can also inspect the cross-validation scores for each combination of parameters:

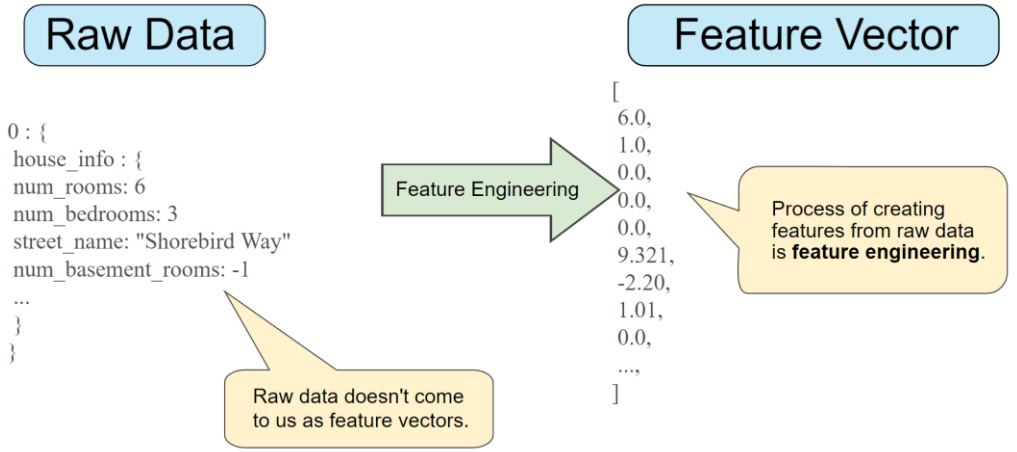
```
df = pd.DataFrame(grid_search.cv_results_)
df
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	param_max_depth	param_min_samples_leaf	params	split0_test_score	split1_test
0	0.001995	1.123916e-07	0.001661	4.700781e-04	gini	1	1	{'criterion': 'gini', 'max_depth': 1, 'min_sam...	0.657895	0.6
1	0.002658	4.694597e-04	0.001662	4.704162e-04	gini	1	2	{'criterion': 'gini', 'max_depth': 1, 'min_sam...	0.657895	0.6
2	0.002992	8.148803e-04	0.002329	4.696327e-04	gini	1	3	{'criterion': 'gini', 'max_depth': 1, 'min_sam...	0.657895	0.6
3	0.003322	4.470465e-04	0.001984	1.168873e-05	gini	1	4	{'criterion': 'gini', 'max_depth': 1, 'min_sam...	0.657895	0.6

Feature Engineering

Northeastern University

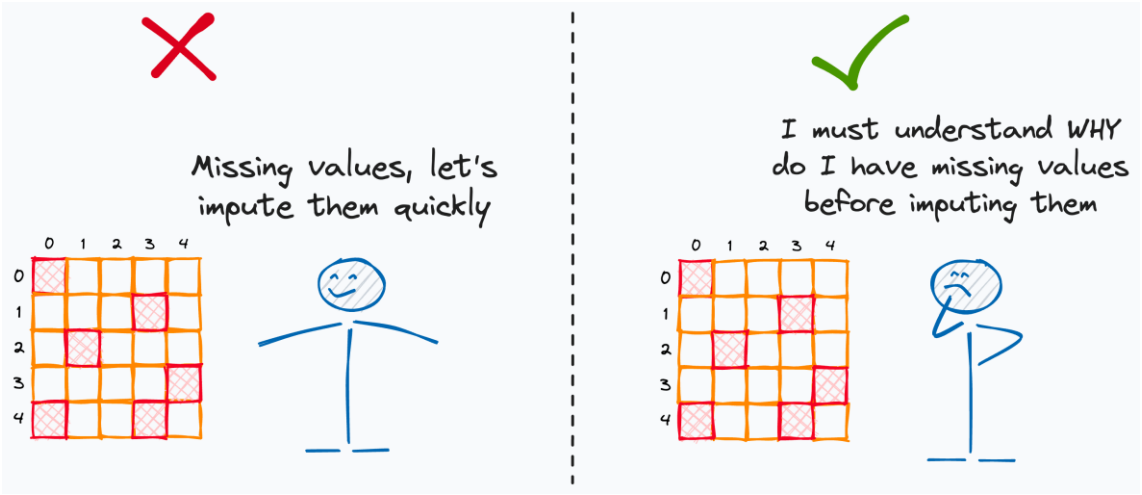
- In the real world, data rarely comes in a tidy, [samples, features] format
- The generation of feature vectors from the raw data is called **feature engineering**



- This process typically involves the following steps:
  - **Data preprocessing:** data preparation, cleaning, and transformation
  - **Feature selection**: selecting the most useful features to train on
  - **Feature extraction**: combining existing features to produce a more useful one

Missing Data

- One of the most common data quality issues in real-world datasets
- Most ML algorithms cannot work with missing features
- Common solutions:
  - Remove samples or features with missing values
  - Imputation: Replace missing values with appropriate fill values such as the mean/median
  - Use model-based techniques to predict the missing value based on other features



Imputation of Missing Data

Northeastern University

- The class **SimpleImputer** is an imputation transformer for completing missing values
  - The **strategy** argument defines the imputation strategy
    - Can be ‘mean’, ‘median’, ‘most_frequent’, or ‘constant’

```
from sklearn.impute import SimpleImputer

imp = SimpleImputer(strategy='mean')
```

```
arr = np.array([[np.nan, 2, 3], [4, np.nan, 6], [10, 5, 9]])
arr
```

```
array([[nan,  2.,  3.],
       [ 4., nan,  6.],
       [10.,  5.,  9.]])
```

```
imp.fit_transform(arr)
```

```
array([[ 7. ,  2. ,  3. ],
       [ 4. ,  3.5,  6. ],
       [10. ,  5. ,  9. ]])
```

Handling Categorical Data

Northeastern University

- A categorical feature represents discrete values that belong to a finite set of categories or classes
- There are two types of categorical variables
  - **Ordinal** features have a natural ordering defined on their values
    - Examples include income level, education level, clothing size, etc.
  - **Nominal** features have no concept of ordering
    - Examples include movie genres, weather names, country names, etc.
- For most ML models categorical features must be first transformed into numbers

Transforming Ordinal Features

Northeastern University

- **OrdinalEncoder** transforms categorical features into integers (ordinal codes)
- The encoding results in a single column of integers [0 to #categories - 1] per feature

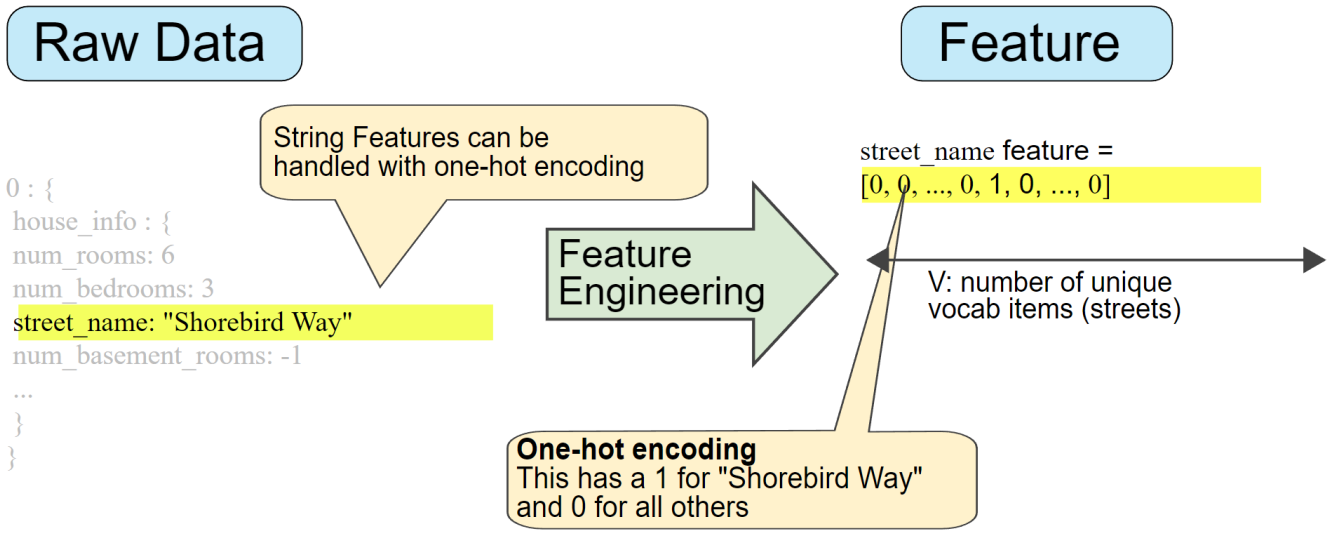
```
from sklearn.preprocessing import OrdinalEncoder

enc = OrdinalEncoder()
```

```
arr = np.array(['MA', 'MA', 'BA', 'PhD', 'BA']).reshape(-1, 1)
enc.fit_transform(arr)
```

```
array([[1.],
       [1.],
       [0.],
       [2.],
       [0.]])
```

One-Hot Encoding

- Convert a categorical feature with *m* labels into a binary vector of size *m* where only one of its elements has 1 and all the other elements have 0



One-Hot Encoding

Northeastern University

- The **OneHotEncoder** class converts categorical values into one-hot vectors

```
from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder(sparse=False)
arr = np.array(['Red', 'Green', 'Red', 'Blue', 'Blue']).reshape(-1, 1)
enc.fit_transform(arr)
```

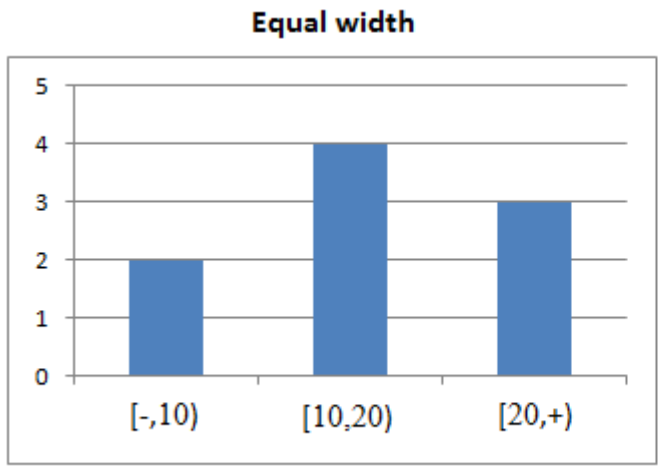
```
array([[0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [1., 0., 0.]])
```

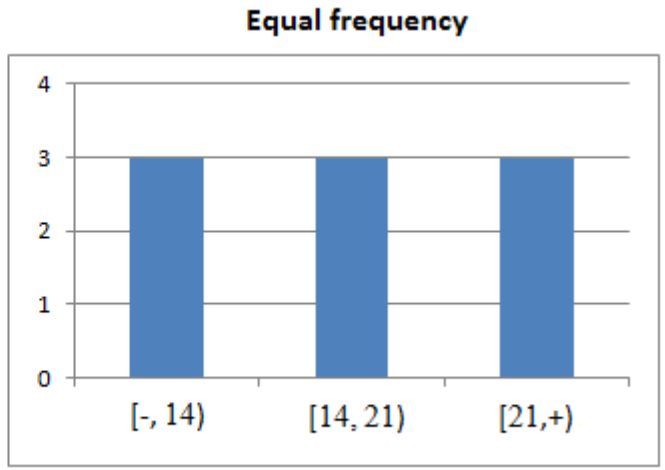
- By default it returns a sparse matrix in which only nonzero values are stored
  - Specify sparse=False to get a dense array

Discretization

- Converting a continuous feature into a discrete one by dividing its range into bins
- Two basic approaches:
  - **Equal-width** – all the bins have the same width
  - **Equal frequency (equal depth)** – all the bins have the same number of objects

- **Data :** 0, 4, 12, 16, 16, 18, 24, 26, 28
- **Equal width**
  - Bin 1: 0, 4 [-,10)
  - Bin 2: 12, 16, 16, 18 [10,20)
  - Bin 3: 24, 26, 28 [20,+)
- **Equal frequency**
  - Bin 1: 0, 4, 12 [-, 14)
  - Bin 2: 16, 16, 18 [14, 21)
  - Bin 3: 24, 26, 28 [21,+)





Discretization

Northeastern University

- **KBinsDiscretizer** can be used to discretize continuous data into intervals
  - **n_bins** defines the number of bins to produce (default is 5)
  - **strategy** defines the strategy used to define the widths of the bins
    - Choose ‘uniform’ for equal-width binning and ‘quantile’ for equal-frequency binning

```
from sklearn.preprocessing import KBinsDiscretizer

x = np.array([0, 4, 12, 16, 16, 18, 24, 26, 28]).reshape(-1, 1)

dis = KBinsDiscretizer(n_bins=3, strategy='uniform', encode='ordinal')
dis.fit_transform(x)
```

```
array([[0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [2.],
       [2.],
       [2.]])
```

Feature Scaling

Northeastern University

- Many ML algorithms struggle when features have different scales
  - Models may become biased toward features having higher magnitude values
- Feature scaling adjust the range of features to ensure they have comparable scales
- Main methods for feature scaling:
  - **Min-max scaling** (or normalization)
    - Rescales features to a specified range, typically [0, 1]

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

  - **Standard scaling** (or standardization)
    - Transforms features to have a mean of 0 and a standard deviation of 1.

$$x' = \frac{x - \overline{x}}{\sigma}$$

Feature Scaling in Scikit-Learn

Northeastern University

- Scikit-Learn provides the classes **MinMaxScaler** and **StandardScaler** for scaling
- Example for using the MinMaxScaler:

```
from sklearn.preprocessing import MinMaxScaler

data = [[1, 1, 1, 1, 1],
        [2, 5, 10, 50, 100],
        [3, 10, 20, 150, 200]]

scaler = MinMaxScaler()
scaler.fit_transform(data)
```

```
array([[0.        , 0.        , 0.        , 0.        , 0.        ],
       [0.5       , 0.44444444, 0.47368421, 0.32885906, 0.49748744],
       [1.        , 1.        , 1.        , 1.        , 1.        ]])
```

Feature Selection

Northeastern University

- Select a subset of the features by removing irrelevant or redundant features
- **Irrelevant features** provide no useful information for the task
  - e.g., the student's ID is irrelevant for predicting the student's GPA
- **Redundant features** duplicate existing information
  - e.g., product purchase price and sales tax amount
- Main methods for feature selection
  - **Filter methods**: Use statistical techniques to rank features by relevance
  - **Embedded methods**: Perform feature selection as part of the model training process
  - **Wrapper methods**: Evaluate feature subsets by training and testing models iteratively
    - e.g., recursive feature elimination (RFE)

Feature Selection in Scikit-Learn

Northeastern University

- The classes in **sklearn.feature_selection** can be used for feature selection
- e.g., **VarianceThreshold** can be used to remove all features with small variance
- To demonstrate, we'll use the Diagnostic Breast Cancer dataset

```
from sklearn.datasets import load_breast_cancer

bc_data = load_breast_cancer()
bc_features = pd.DataFrame(bc_data.data,
                           columns=bc_data.feature_names)
bc_features.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980

5 rows × 30 columns

Feature Selection in Scikit-Learn

Northeastern University

- We can remove features with less than 0.05 variance:

```
from sklearn.feature_selection import VarianceThreshold

vt = VarianceThreshold(threshold=0.05)
vt.fit(bc_features)
```

- To view the variances as well as which features were selected by this algorithm, we can use the variances_ property and the get_support(...) method, respectively:

```
pd.DataFrame({'variance': vt.variances_,
              'select_feature': vt.get_support()},
             index=bc_features.columns).T
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	fi dimei
variance	12.3971	18.4664	589.403	123626	0.000197452	0.00278429	0.00634408	0.00150301	0.000750222	4.97
select_feature	True	True	True	True	False	False	False	False	False	

2 rows × 30 columns

Feature Extraction

Northeastern University

- Create new features from the existing ones to better capture the patterns in the data
- Often requires innovation and domain knowledge from the data analyst
- For example, assume we have the following data of air pollution by country:

```
pollution = pd.DataFrame([
    {'country': 'New Zealand', 'area': 269190, 'population': 4705818, 'air_pollution': 5},
    {'country': 'Qatar', 'area': 11437, 'population': 2639211, 'air_pollution': 103},
    {'country': 'India', 'area': 2639211, 'population': 1339180127, 'air_pollution': 65},
    {'country': 'Spain', 'area': 504781, 'population': 46354321, 'air_pollution': 9},
    {'country': 'Cameroon', 'area': 475440, 'population': 24053727, 'air_pollution': 62},
    {'country': 'Thailand', 'area': 513120, 'population': 69222885, 'air_pollution': 27},
    {'country': 'Egypt', 'area': 1001449, 'population': 99887475, 'air_pollution': 75}
], columns=['country', 'area', 'population', 'air_pollution'])
pollution = pollution.set_index('country')
pollution
```

country	area	population	air_pollution
New Zealand	269190	4705818	5
Qatar	11437	2639211	103
India	2639211	1339180127	65
Spain	504781	46354321	9
Cameroon	475440	24053727	62
Thailand	513120	69222885	27
Egypt	1001449	99887475	75

```
pollution.corr()['air_pollution']
```

```
area             0.153068
population       0.187176
air_pollution    1.000000
Name: air_pollution, dtype: float64
```

Feature Extraction

Northeastern University

- Let's add a population density feature:

```
pollution['density'] = pollution['population'] / pollution['area']
```

- And now examine the correlation coefficients again:

```
pollution.corr()['air_pollution']
```

```
area             0.153068
population       0.187176
air_pollution    1.000000
density          0.435159
Name: air_pollution, dtype: float64
```

  - There is a relatively high correlation between density and air_pollution