Repor t

제목 : Machin Learnig with pthon

머신 러닝 예제를 통한 파이썬 라이브러리 탐구 및 알고리즘 분석

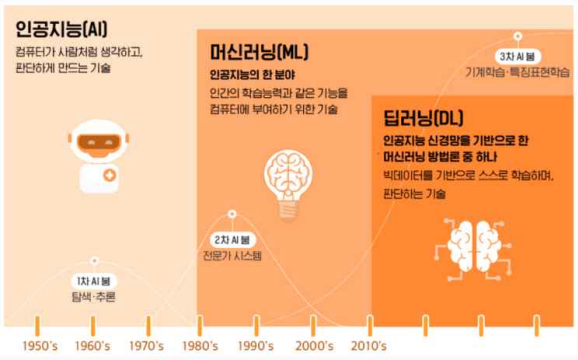
학 과	스마트팩토리과
실습조	A반 1조
학번	2217110229
이름	임신홍
과목명	AI 제어 실습
제출일	2022.12.13

목차

1. 머신 러닝이란

1.1. 머신 러닝 정의

머신러닝은 컴퓨터를 인간처럼 학습시킴으로써 컴퓨터가 새로운 규칙을 생성할 수 있지 않을까 하는 시도에서 시작되어 컴퓨터가 스스로 학습할 수 있도록 도와주는 알고리즘이나 기술을 개발하는 분야를 말합니다.

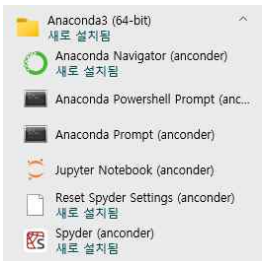


붓꽃 예제를 참고하는 책인 'Machin Learnig with pthon' 에서는 데이터에서 지식을 추출하는 작업이라 정의하였습니다.
초기의 지능형 로직은 if else 명령을 사용하여 결정 규칙을 수동으로 만들었습니다.
이는 작업이 조금만 변경이 오더라도 전체 시스템을 다시 개발해야 한다는 단점이 있었습니다. 머신러닝 알고리즘은 이미 알려진 사례를 바탕으로 일반화된 모델을 만들어 의사결정 프로세스를 자동화하는 것들입니다. 이런 방식을 지도학습이라고 하며, 컴퓨터가 인식할 수 있는 형태의 입력 데이터를 받으면 기대되는 출력을 전체를 고치지 않고도 출력 가능합니다.

1.2. 이용 프로그램과 라이브러리

1.2.1 주피터 노트북

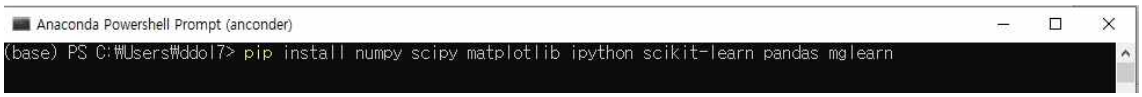
주피터 노트북은 프로그램 코드를 브라우저에서 실행해주는 대화식 환경입니다. 이런 방식은 탐색적 데이터 분석에 아주 적합하여 많은 데이터 분석가가 주피터 노트북을 사용하고 있습니다. 아나콘다를 설치하면 쉽게 주피터 노트북을 사용할수 있습니다.



1.2.2 이용 파이썬 라이브러리

아나콘다 설치 후 Powershell에 아래 문장을 칩니다. 여기서 설치되는 패키지들이 붓꽃 예제에 사용됩니다.

$ pip install numpy scipy matplotlib ipython scikit-learn pandas mglearn



NumPy(http://www.numpy.org/)는 파이썬으로 과학 계산을 하려면 꼭 필요한 패키지입니다. 다차원 배열을 위한 기능과 선형 대수 연산과 푸리에 변환 같은 고수준 수학 함수와 유사난수 생성기를 포함합니다.

SciPy(https://www.scipy.org/scipylib)는 과학 계산용 함수를 모아놓은 파이썬 패키지입니다. SciPy는 고성능 선형대수, 함수 최적화, 신호 처리, 특수한 수학 함수와 통계 분포 등을 포함한 많은 기능을 제공합니다.

matplotlib(https://matplotlib.org/)은 파이썬의 대표적인 과학 계산용 그래프 라이브러리입니다. 선 그래프, 히스토그램, 산점도 등을 지원하며 출판에 쓸 수 있을 만큼의 고품질 그래프를 그려줍니다.

Ipython(http://ipython.org/ipython-doc/3/overview.html)은 Interactive python으로 대화형 방식의 분석 및 개발을 목적으로 만들었다. 이는 주피터 노트북의 파이썬 사용에 필요합니다.

pandas (http://pandas.pydata.org/)는 데이터 처리와 분석을 위한 파이썬 라이브러리입니다. R의 data.frame을 본떠서 설계한 DataFrame이라는 데이터 구조를 기반으로 만들어졌습니다.

mglearn는 그래프나 데이터 적재와 관련한 세세한 코드를 일일이 쓰지 않아도 되게끔 만든 유틸리티 함수들입니다.

scikit-learn(http://scikit-learn.org/stable/user_guide.html)는 오픈소스로 자유롭게 사용하거나 배포할 수 있고, 누구나 소스 코드를 보고 실제로 어떻게 동작하는지 쉽게 확인할 수 있습니다. NumPy와 pandas를 이용해 입력 데이터를 기본 데이터 배열에 맞게 정렬, SciPy로 계산 matplotlib를 통한 그래프 작성, 코드를 편하게 만들기위한 mglearn등을 사용하여 이 오픈소스를 동작시킵니다. 대화식으로 개발하려면 IPython과 주피터 노트북도 설치해야 합니다.

2. 붓꽃 품종 분류 머신 러닝

2.1. 예제 코드

In [2]:
```
import sys
print("Python version:", sys.version)

import pandas as pd
print("pandas version:", pd.__version__)

import matplotlib
print("matplotlib version:", matplotlib.__version__)

import numpy as np
print("NumPy version:", np.__version__)

import scipy as sp
print("SciPy version:", sp.__version__)

import IPython
print("IPython version:", IPython.__version__)

import sklearn
print("scikit-learn version:", sklearn.__version__)

import mglearn
```

```
Python version: 3.9.13 (main, Aug 25 2022, 23:51:50) [MSC v.1916 64 bit (AMD64)]
pandas version: 1.4.4
matplotlib version: 3.5.2
NumPy version: 1.21.5
SciPy version: 1.9.1
IPython version: 7.31.1
scikit-learn version: 1.0.2
```

In [3]:
```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
```

In [4]:
```
print("Keys of iris_dataset:₩n", iris_dataset.keys())
```

```
Keys of iris_dataset:
 dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
```

In [5]:
```
print(iris_dataset['DESCR'][:193] + "₩n...")
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, pre
...
```

In [6]:
```
print("타겟이름:", iris_dataset['target_names'])
```

```
타겟이름: ['setosa' 'versicolor' 'virginica']
```

In [7]:
```
print("특성이름:₩n", iris_dataset['feature_names'])
```

```
특성이름:
 ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

In [8]:
```
print("데이타 타입:", type(iris_dataset['data']))
```

```
데이타 타입: <class 'numpy.ndarray'>
```

In [9]:
```
print("데이타 크기:", iris_dataset['data'].shape)
```

```
데이타 크기: (150, 4)
```

In [10]:
```
print("처음 다섯번째 행 데이타:₩n", iris_dataset['data'][:5])
```

```
처음 다섯번째 행 데이타:
 [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
```

In [11]:
```
print("타겟타입:", type(iris_dataset['target']))
```

```
타겟타입: <class 'numpy.ndarray'>
```

In [12]:
```
print("타겟크기:", iris_dataset['target'].shape)
```

```
타겟크기: (150,)
```

In [13]:
```
print("Target:₩n", iris_dataset['target'])
```

```
Target:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

In [14]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    iris_dataset['data'], iris_dataset['target'], random_state=0)
```

In [15]:
```
print("X_train 크기:", X_train.shape)
print("y_train 크기:", y_train.shape)
```

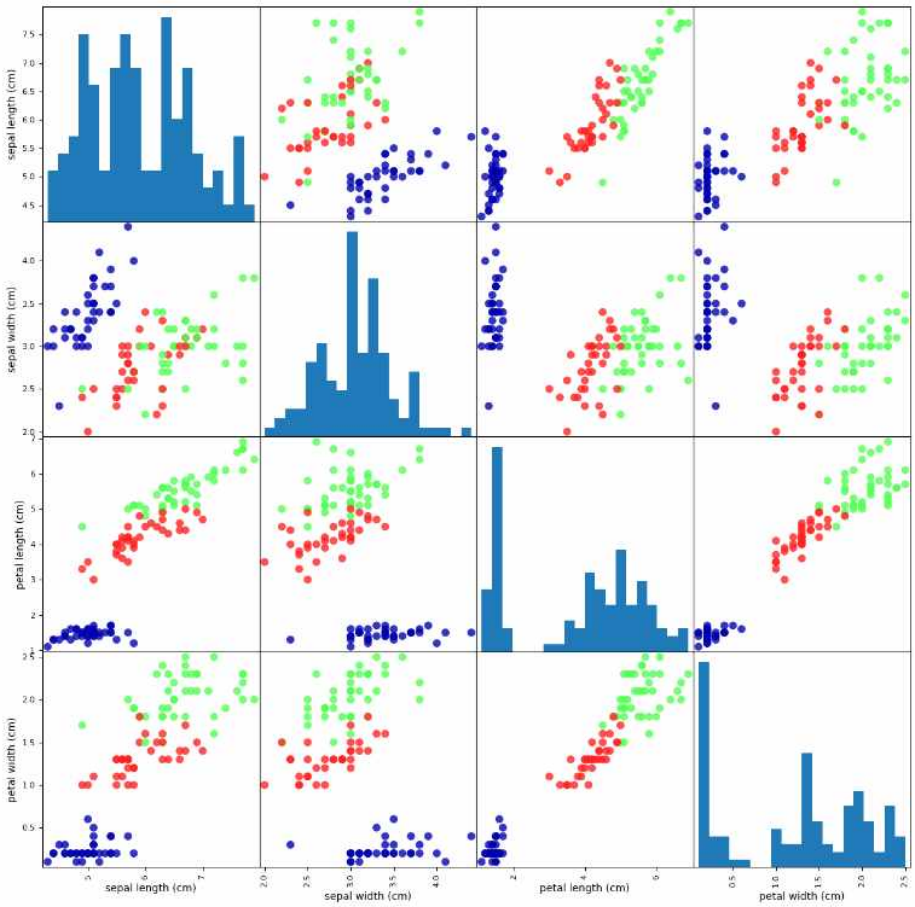
```
X_train 크기: (112, 4)
y_train 크기: (112,)
```

In [16]:
```
print("X_test 크기:", X_test.shape)
print("y_test 크기:", y_test.shape)
```

```
X_test 크기: (38, 4)
y_test 크기: (38,)
```

In [17]:
```
# create dataframe from data in X_train
# label the columns using the strings in iris_dataset.feature_names
iris_dataframe = pd.DataFrame(X_train, columns=iris_dataset.feature_names)
# create a scatter matrix from the dataframe, color by y_train
pd.plotting.scatter_matrix(iris_dataframe, c=y_train, figsize=(15, 15),
                           marker='o', hist_kwds={'bins': 20}, s=60,
                           alpha=.8, cmap=mglearn.cm3)
```

Out[17]:
```
array([[<AxesSubplot:xlabel='sepal length (cm)', ylabel='sepal length (cm)'>,
        <AxesSubplot:xlabel='sepal width (cm)', ylabel='sepal length (cm)'>,
        <AxesSubplot:xlabel='petal length (cm)', ylabel='sepal length (cm)'>,
        <AxesSubplot:xlabel='petal width (cm)', ylabel='sepal length (cm)'>],
       [<AxesSubplot:xlabel='sepal length (cm)', ylabel='sepal width (cm)'>,
        <AxesSubplot:xlabel='sepal width (cm)', ylabel='sepal width (cm)'>,
        <AxesSubplot:xlabel='petal length (cm)', ylabel='sepal width (cm)'>,
        <AxesSubplot:xlabel='petal width (cm)', ylabel='sepal width (cm)'>],
       [<AxesSubplot:xlabel='sepal length (cm)', ylabel='petal length (cm)'>,
        <AxesSubplot:xlabel='sepal width (cm)', ylabel='petal length (cm)'>,
        <AxesSubplot:xlabel='petal length (cm)', ylabel='petal length (cm)'>,
        <AxesSubplot:xlabel='petal width (cm)', ylabel='petal length (cm)'>],
       [<AxesSubplot:xlabel='sepal length (cm)', ylabel='petal width (cm)'>,
        <AxesSubplot:xlabel='sepal width (cm)', ylabel='petal width (cm)'>,
        <AxesSubplot:xlabel='petal length (cm)', ylabel='petal width (cm)'>,
        <AxesSubplot:xlabel='petal width (cm)', ylabel='petal width (cm)'>]],
      dtype=object)
```



In [18]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
```

In [19]:
```
knn.fit(X_train, y_train)
```

Out[19]:
```
KNeighborsClassifier(n_neighbors=1)
```

In [20]:
```
X_new = np.array([[5, 2.9, 1, 0.2]])
print("X_new.shape:", X_new.shape)
```

```
X_new.shape: (1, 4)
```

In [21]:
```
prediction = knn.predict(X_new)
print("Prediction:", prediction)
print("Predicted target name:",
       iris_dataset['target_names'][prediction])
```

```
Prediction: [0]
Predicted target name: ['setosa']
```

In [22]:
```
y_pred = knn.predict(X_test)
print("Test set predictions:₩n", y_pred)
```

```
Test set predictions:
 [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
 2]
```

In [23]:
```
print("Test set score: {:.2f}".format(np.mean(y_pred == y_test)))
```

```
Test set score: 0.97
```

In [24]:
```
print("Test set score: {:.2f}".format(knn.score(X_test, y_test)))
```

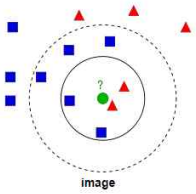
```
Test set score: 0.97
```

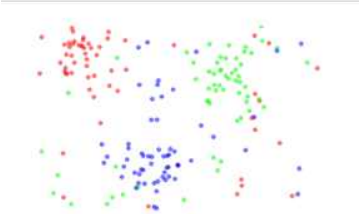
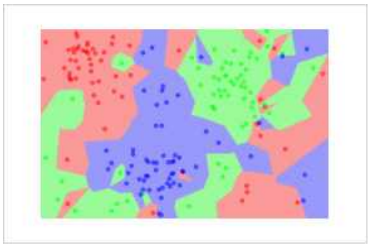
In [25]:
```
X_train, X_test, y_train, y_test = train_test_split(
    iris_dataset['data'], iris_dataset['target'], random_state=0)

knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)

print("Test set score: {:.2f}".format(knn.score(X_test, y_test)))
```

```
Test set score: 0.97
```

2.2. 사용된 알고리즘 분석 - k-최근접 이웃 알고리즘(k-Nearest Neighbor)

k-최근접 이웃 알고리즘(k-Nearest Neighbor)이란?
특징 공간에서 테스트 데이터와 가장 가까이 있는 k개의 학습 데이터를 찾아 분류 또는 회귀를 수행하는 지도 학습 알고리즘의 하나입니다.



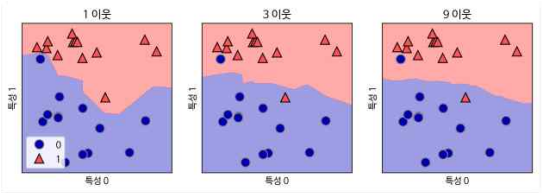



학습 데이터가 녹색이라고 가정하였을 때 원래 데이터(파랑색, 빨강색) 중 가장 가까지 있는 데이터의 클래스를 부여합니다. 위 사진에서 녹색 데이터는 빨강색 데이터와 가까이 있지만 빨강색 데이터라고 판단하기에는 애매합니다. 이처럼 잘못 분류될 가능성이 있는데 이를 해결하기 위해 가장 가까이 있는 것을 k개 찾습니다. 선별한 k개 데이터에서 파랑색, 빨강색 데이터 개수를 세어 더 많이 선별된 데이터의 클래스를 부여하게 됩니다. 예를 들어 5개 데이터를 선별했을 때 3개가 파랑색, 2개가 빨강색이면 파랑색 클래스를 부여합니다.

scikit-learn 라이브러리의 k-최근접 이웃 알고리즘(k-Nearest Neighbor)

```
In [18]: from sklearn.neighbors import KNeighborsClassifier
         knn = KNeighborsClassifier(n_neighbors=1)
```

KNeighborsClassifier 분석
2차원 데이터셋이므로 가능한 모든 테스트 포인트의 예측을 xy 평면에 그려볼 수 있습니다. 그리고 각 데이터 포인트가 속한 클래스에 따라 평면에 색을 칠합니다. 이렇게 하면 알고리즘이 클래스 0과 클래스 1로 지정한 영역으로 나뉘는 결정 경계decision boundary를 볼 수 있습니다. 다음 코드는 이웃이 하나, 셋, 아홉 개일 때의 결정 경계를 보여줍니다.



이웃을 하나 선택했을 때는 결정 경계가 훈련 데이터에 가깝게 따라가고 있습니다.
이웃의 수를 늘릴수록 결정 경계는 더 부드러워집니다. 부드러운 경계는 더 단순한 모델을 의미합니다. 다시 말해 이웃을 적게 사용하면 모델의 복잡도가 높아지고 많이 사용하면 복잡도는 낮아집니다. 훈련데이터 전체 개수를 이웃의 수로 지정하는 극단적인 경우에는 모든 테스트 포인트가 같은 이웃(모든 훈련데이터)을 가지게 되므로 테스트 포인트에 대한 예측은 모두 같은 값이 됩니다. 즉 훈련 세트에서 가장 많은 데이터 포인트를 가진 클래스가 예측값이 됩니다.

Open CV2 라이브러리의 k-최근접 이웃 알고리즘(k-Nearest Neighbor)

cv2.ml.KNearest_create() -> retval

• retval: cv2.ml_KNearest 객체

객체 생성 코드

객체를 생성하면 k=10 디폴트 값으로 설정되어 있습니다.
k 값이 너무 많아도 일반화 되어 변별력이 떨어지게 됩니다.
k는 적당한 값을 지정해줘야 하는데 .findNearest 함수로 k 값을 지정할 수 있습니다.

cv2.ml_KNearest.train(samples, layout, responses) -> retval

• samples: 학습 데이터 행렬. numpy.ndarray. shape=(N, d), dtype=numpy.float32.

• layout: 학습 데이터 배치 방법.
cv2.ROW_SAMPLE : 하나의 데이터가 한 행으로 구성됨
cv2.COL_SAMPLE : 하나의 데이터가 한 열로 구성됨

• responses: 각 학습 데이터에 대응되는 응답(레이블) 행렬. numpy.ndarray. shape=(N, 1), dtype=numpy.int32 또는 numpy.float32.

• retval: 학습이 성공하면 True.

KNN 알고리즘 학습

cv.ml_KNearest.findNearest(samples, k, results=None, neighborResponses=None, dist=None , flags=None) -> retval, results, neighborResponses, dist

• samples: 입력 벡터가 행 단위로 저장된 입력 샘플 행렬. numpy.ndarray. shape=(N, d), dtype=numpy.float32.

• k: 사용할 최근접 이웃 개수

• results: 각 입력 샘플에 대한 예측(분류 또는 회귀) 결과를 저장한 행렬. numpy.ndarray. shape=(N, 1), dtype=numpy.float32.

• neighborResponses: 예측에 사용된 k개의 최근접 이웃 클래스 정보 행렬. numpy.ndarray. shape=(N, k), dtype=numpy.float32.

• dist: 입력 벡터와 예측에 사용된 k개의 최근접 이웃과의 거리를 저장한 행렬. numpy.ndarray. shape=(N, k), dtype=numpy.float32.

• retval: 입력 벡터가 하나인 경우에 대한 응답

KNN 알고리즘으로 입력 데이터의 클래스 예측

.findNearest 함수는 neighborResponses와 dist 값을 추가로 받을 수 있습니다.
neighborResponses는 근접 데이터 정보를 갖고 있습니다.
dist는 L2 norm 정보를 제공합니다.