201600779 김영민

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('ggplot')
from tqdm import tqdm
from sklearn.datasets import load_iris
# import random
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
iris = load_iris()
input_data = iris.data
target = iris.target
df = pd.DataFrame(input_data,columns = iris.feature_names)
df['Target'] = target
```

In [3]:

```
np.random.seed(42)
# random.seed(42)
```

In [4]:

```
class neuralNetwork:

    #신경망 초기화하기
    def __init__(self,inputnodes,hiddennodes,outputnodes,learningrate):
        self.inodes=inputnodes
        self.hnodes=hiddennodes
        self.onodes=outputnodes

        '''
        가중치 행렬 wih와 who
        배열 내 가중치는 w_i_j로 표기, 노드 i에서 다음 계층의 노드 j로 연결됨을 의미
        w11 w21
        w12 w22 등'''
        self.wih=np.random.normal(0.0,pow(self.hnodes,-0.5),(self.hnodes,self.inodes))   #정규분포로
        self.who=np.random.normal(0.0,pow(self.onodes,-0.5),(self.onodes,self.hnodes))

        #학습률
        self.lr=learningrate

        #활성화 함수로 시그모이드 함수를 이용
        def sigmoid(x):
            return 1/(1+np.exp(-x))



        self.activation_function = lambda x: sigmoid(x)

        pass
    #신경망 학습시키기
    def train(self, inputs_list,targets_list):
        #입력 리스트를 2차원의 행렬로 반환
        inputs=np.array(inputs_list,ndmin=2).T
        targets=np.array(targets_list,ndmin=2).T

        #은닉 계층으로 들어오는 신호를 계산
        hidden_inputs=np.dot(self.wih,inputs)
        #은닉 계층에서 나가는 신호를 계산
        hidden_outputs=self.activation_function(hidden_inputs)
        #최종 출력 계층으로 들어오는 신호를 계산
        final_inputs=np.dot(self.who,hidden_outputs)
        #최종 출력 계층에서 나가는 신호를 계산
        final_outputs=self.activation_function(final_inputs)

        #출력 계층의 오차는(실제값-계산값)
        output_errors = targets-final_outputs
        #은닉 계층의 오차는 가중치에 의해 나뉜 출력 계층의 오차들을 재조합해서 계산
        hidden_errors = np.dot(self.who.T,output_errors)

        #은닉 계층과 출력 계층 간의 가중치 업데이트
        self.who+= self.lr*np.dot((output_errors*final_outputs*(1.0-final_outputs)),np.transpose(hi

        #입력 계층과 은닉 계층 간의 가중치 업데이트
        self.wih+= self.lr*np.dot((hidden_errors*hidden_outputs*(1.0-hidden_outputs)),np.transpose(

        pass

    #신경망 질의하기
    def query(self, input_list):
```

```
        #입력 리스트를 2차원 행렬로 변환
        inputs=np.array(input_list, ndmin=2).T

        #은닉 계층에서 들어오는 신호를 계산
        hidden_inputs=np.dot(self.wih,inputs)
        #은닉 계층에서 나가는 신호를 계산
        hidden_outputs=self.activation_function(hidden_inputs)
        #최종 출력 계층으로 들어오는 신호를 계산
        final_inputs=np.dot(self.who,hidden_outputs)
        #최종 출력 계층에서 나가는 신호를 계산
        final_outputs=self.activation_function(final_inputs)

        return final_outputs
```

자동 추출

In [5]:

```
from sklearn.model_selection import train_test_split
train,test = train_test_split(df,test_size = .2,random_state = 42)
```

In [6]:

```
train_x = train.drop('Target',axis=1).values
train_y = train.Target.values
test_x = test.drop('Target',axis = 1).values
test_y = test.Target.values
```

In [7]:

```
input_layer = 4
hidden_layer = 10
outuput_layer = 3
lr = 0.1
```

In [8]:

```
n = neuralNetwork(input_layer,hidden_layer,outuput_layer,lr)
epochs = 300
for e in tqdm(range(epochs)):
    for i in range(len(train_x)):
        inp = train_x[i]
        tar = train_y[i]
        n.train(inp,tar)
scorecard = []
for i in range(len(test_x)):
    q = test_x[i]
    a = test_y[i]
    outputs = n.query(q)
    label=np.argmax(outputs)
    if label == a:
        scorecard.append(1)
    else:
        scorecard.append(0)
score = np.sum(scorecard)/len(scorecard)
print('Performance : %.2f'%score)
```

```
100%|██████████████████████████████████████████████████████████████████████████████
| 300/300 [00:00<00:00, 336.09it/s]

Performance : 0.63
```

수동 추출

In [9]:

```
zero = df[df['Target'] == 0][:40]
one = df[df['Target'] == 1][:40]
two = df[df['Target'] == 2][:40]

train_df = []
train_df.append(zero)
train_df.append(one)
train_df.append(two)
train = pd.concat(train_df)
```

In [10]:

```
zero = df[df['Target'] == 0][40:]
one = df[df['Target'] == 1][40:]
two = df[df['Target'] == 2][40:]

test_df = []
test_df.append(zero)
test_df.append(one)
test_df.append(two)
test = pd.concat(test_df)
```

In [11]:

```
train_x = train.drop('Target',axis=1).values
train_y = train.Target.values
test_x = test.drop('Target',axis=1).values
test_y = test.Target.values
```

In [12]:

```
n = neuralNetwork(input_layer,hidden_layer,outuput_layer,lr)
epochs = 300
for e in tqdm(range(epochs)):
    for i in range(len(train_x)):
        inp = train_x[i]
        tar = train_y[i]
        n.train(inp,tar)
scorecard = []
for i in range(len(test_x)):
    q = test_x[i]
    a = test_y[i]
    outputs = n.query(q)
    label=np.argmax(outputs)
    if label == a:
        scorecard.append(1)
    else:
        scorecard.append(0)
score = np.sum(scorecard)/len(scorecard)
print('Performance : %.2f'%score)
```

```
100%|██████████████████████████████████████████████████████████████████████████████
| 300/300 [00:00<00:00, 321.84it/s]

Performance : 0.60
```

In [13]:

```
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
```

In [14]:

```
rf = RandomForestClassifier(random_state = 42)
xgb = XGBClassifier(random_state=42)
rf.fit(train_x,train_y)
rf_pred = rf.predict(test_x)
rf_acc = accuracy_score(rf_pred,test_y)
xgb.fit(train_x,train_y)
xgb_pred = xgb.predict(test_x)
xgb_acc = accuracy_score(xgb_pred,test_y)
print('==========='*8)
print('RF score %.2f'%rf_acc)
print('XGB score %.2f'%xgb_acc)
```

```
[19:10:53] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/sr
c/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used wit
h the objective 'multi:softprob' was changed from 'merror' to 'mlogloss'. Explicitly
set eval_metric if you'd like to restore the old behavior.
============================================================================================
========================================
RF score 1.00
XGB score 1.00
```

201600779 김영민