<CSE 302 Assignment 2>

202011212 진호진

먼저, 코드는 과제 설명에 적혀있는 대로, 다른 library는 거의 쓰지 않고, Scikit-learn library를 활용하여 4가지 방법을 실행해 보았다. 각각의 방법을 인자를 바꾸어 가며 실행 할 때, 수업자료와 「https://scikit-learn.org/stable/index.html」을 활용하여 어떤 인자가 있고, 어떤 옵션을 적용할 수 있을지 확실하게 check 하는 과정을 거쳤다.

1. Dataset 준비 과정

mnist를 로드 해주고, dataset을 traning set 과 test set으로 나누는 과정을 진행했다.

```
split_ratio = 0.9
n_train = int(digits.data.shape[0] * split_ratio)

n_test = digits.data.shape[0] - n_train

X_train = digits.data[:n_train]
Y_train = digits.target[:n_train]

X_test = digits.data[n_train:]
Y_test = digits.target[n_train:]
```

2. Logistic regression models

Logistic regression model의 option으로는 크게 penalty와 C가 있었다. 따라서 각각의 penalty에 대하여 C값을 변경해 주면서 accuracy 값을 비교해 보았다. (solver는 모든 penalty에 상관없는 saga를 이용하였다.)

```
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
C_value=[0.001,0.01,0.1,1,10]
for i in C_value:
    model = LogisticRegression(penalty='l1',C=i, solver = 'saga')
    model.fit(X_train, Y_train)
    prediction = model.predict(X_test)
    print(accuracy_score(Y_test, prediction))
```

penalty가 l1인 경우, C값을 0.001에서 10까지 10을 곱해가며 적용

```
C_value=[0.001,0.01,0.1,1,10]
for i in C_value:
    model = LogisticRegression(penalty='l2',C=i, solver = 'saga')
    model.fit(X_train, Y_train)
    prediction = model.predict(X_test)
    print(accuracy_score(Y_test, prediction))
```

penalty가 l2인 경우, C값을 0.001에서 10까지 10을 곱해가며 적용

	0.001	0.01	0.1	1	10
L1	0.55555555	0.872222222	0.92222222	0.933333333	0.933333333
L2	0.92777777	0.933333333	0.933333333	0.933333333	0.933333333

각각 코드의 출력 결과를 표로 나타내면 다음과 같다.
이 결과를 보고 가장 높은 accuracy를 갖는 model은 색칠된 option을 적용했을 때라고 유추할 수 있다.

3. K-NN classifiers

K-NN classifiers에서 option으로는 neighbor을 지정해 줄 수 있었다. 따라서 neighbor 값을 변경하면서 accuracy를 관찰하였다.

```
from sklearn.neighbors import KNeighborsClassifier

neighbor = [1,2,3,4,5,6,7,8,9,10]

for i in neighbor:
    model = KNeighborsClassifier(n_neighbors = i)
    model.fit(X_train, Y_train)
    prediction = model.predict(X_test)
    print(accuracy_score(Y_test, prediction))
```

neighbor 값을 바꿔가면서 accuracy 값을 출력한다

1	2	3	4	5
0.9666666666	0.977777777	0.9666666666	0.9666666666	0.9611111111
6	7	8	9	10
0.9611111111	0.9611111111	0.9611111111	0.9611111111	0.9611111111

코드의 출력 결과를 표로 나타내면 다음과 같다.
이 결과를 보고 neighbor가 2일 때 가장 높은 accuracy를 가지는 것을 알 수 있다.

4. SVM classifiers

SVM classifiers에서는 option으로 먼저 kernel을 선택 할 수 있었고, poly인 경우에는 차수인 degree, rbf인 경우에는 gamma 값을 변경할 수 있었다.

```
from sklearn.svm import SVC
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

kernel = ["linear","poly","rbf","sigmoid"]

for i in kernel:
    model = make_pipeline(StandardScaler(), SVC(kernel= i , gamma='auto'))
    model.fit(X_train, Y_train)
    prediction = model.predict(X_test)
    print(accuracy_score(Y_test, prediction))
```

kernel을 변경하면서 accuracy 값을 출력한다.

linear	poly	rbf	sigmoid
0.9611111111	0.89444444444	0.94444444444	0.92777777777

```
degree = [1,2,3,4,5]

for i in degree:
    model = make_pipeline(StandardScaler(), SVC(kernel= "poly" , degree=i ,gamma='auto'))
    model.fit(X_train, Y_train)
    prediction = model.predict(X_test)
    print(accuracy_score(Y_test, prediction))
```

kernel이 poly인 경우 그 차수인 degree 값을 변경하면서 accuracy 출력

1	2	3	4	5
0.944444444	0.9444444444	0.894444444	0.7	0.5611111111

```
gamma = [0.001,0.01,0.1,1]

for i in gamma:
    model = make_pipeline(StandardScaler(), SVC(kernel= "rbf" , gamma=i))
    model.fit(X_train, Y_train)
    prediction = model.predict(X_test)
    print(accuracy_score(Y_test, prediction))
```

kernel이 rbf인 경우 gamma 값을 변경하면서 accuracy 출력

0.001	0.01	0.1	1
0.9166666666	0.9444444444	0.8833333333	0.12777777777

kernel의 경우 linear일 때 가장 높은 accuracy를 보였고, poly인 경우 degree가 1,2차, rbf의 경우 gamma값이 0.01일때 최고의 accuracy를 기록하고 그 이후로는 현저히 떨어지는 것을 볼 수 있었다.

5. Random forest classifiers

Random forest classifiers는 n_estimators를 변경하며 accuracy를 관찰 해 보았다.

```
from sklearn.ensemble import RandomForestClassifier
n_estimators = [10, 50, 100, 500]
for i in n_estimators:
    model = RandomForestClassifier(n_estimators=i)
    model.fit(X_train, Y_train)
    prediction = model.predict(X_test)
    print(accuracy_score(Y_test, prediction))
```

n_estimator를 변경하며 accuracy 출력

10	50	100	500
0.8833333333	0.9277777777	0.938888888888	0.938888888888

n_estimator가 100, 500인 경우 가장 높은 accuracy를 보였다. 이 경향이 지속되는지 알기 위해 1000을 넣어서 비교를 해보았다. 1000 또한 비슷한 값인 0.93888888로 출력되어 n_estimator가 커지면 정확도가 증가한다는 경향을 얻을 수 있었다.

6. Best result table

	Logistic regression	K-NN	SVM	Random Forest
Accuracy	0.933333333	0.97777777	0.96111111	0.938888888888