과제명	HW2	학번	201611182	이름	최동현

보고서 내용	주요코드, 결과, 결과에 대한 분석을 기술한 보고서(jupyter notebook 활용)

문제 번호	내용

data set 불러 오기	○ 주요 코드
	`# Mnist data set 불러오기`
	`import tensorflow as tf`
	`from tensorflow import keras`
	`from tensorflow.keras import layers`
	`mnist = tf.keras.datasets.mnist`
	`(X_train, y_train), (X_test, y_test) = mnist.load_data()`
	`X_train, X_test = X_train / 255.0, X_test / 255.0`
	`print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)`
	`nsamples, nx, ny = X_train.shape`
	`X_train = X_train.reshape((nsamples,nx*ny))`
	`nsamples, nx, ny = X_test.shape`
	`X_test = X_test.reshape((nsamples,nx*ny))`
	`print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)`
	`(60000, 28, 28) (60000,) (10000, 28, 28) (10000,)`
	`(60000, 784) (60000,) (10000, 784) (10000,)`
	-jupyter notebook 로컬 환경에서 Mnist data set을 불러온 결과.
	-train, test data의 shape을 확인한 결과.

1번	○ LogisticRegression 기본 모델
	`# LogisticRegression`
	`from sklearn.model_selection import train_test_split`
	`from sklearn.linear_model import LogisticRegression`
	`L_Reg = LogisticRegression().fit(X_train, y_train)`
	`print("Training set score: {:.4f}".format(L_Reg.score(X_train, y_train)))`
	`print("Test set score: {:.4f}".format(L_Reg.score(X_test, y_test)))`
	`c:₩users₩최동현₩appdata₩local₩programs₩python₩python38₩lib₩site-packages₩sklearn₩linear_model₩_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):`
	`STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.`
	`Increase the number of iterations (max_iter) or scale the data as shown in:`
	`https://scikit-learn.org/stable/modules/preprocessing.html`
	`Please also refer to the documentation for alternative solver options:`
	`https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression`
	`n_iter_i = _check_optimize_result(`
	`Training set score: 0.9351`
	`Test set score: 0.9258`
	-LogisticRegression model의 기본 매개변수를 통해 학습한 결과.
	-Test set score : 0.9258

○ LogisticRegression model trial

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt

for C in [0.001, 1, 100]:
    lr_l2 = LogisticRegression(solver='lbfgs', C=C, penalty="l2", max_iter=500).fit(X_train, y_train)
    print("C={:.3f} 인 l2 로지스틱 회귀의 훈련 정확도: {:.4f}".format(
          C, lr_l1.score(X_train, y_train)))
    print("C={:.3f} 인 l2 로지스틱 회귀의 테스트 정확도: {:.4f}".format(
          C, lr_l1.score(X_test, y_test)))
```

```
C=0.001 인 l2 로지스틱 회귀의 훈련 정확도: 0.9421
C=0.001 인 l2 로지스틱 회귀의 테스트 정확도: 0.9228
```

```
c:₩users₩최동현₩appdata₩local₩programs₩python₩python38₩lib₩site-packages₩sklearn₩linear_model₩_logistic.py:814: ConvergenceWarning: lbf
gs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
C=1.000 인 l2 로지스틱 회귀의 훈련 정확도: 0.9421
C=1.000 인 l2 로지스틱 회귀의 테스트 정확도: 0.9228
```

-LogisticRegression model의 lbfgs와 l2규제를 적용한 결과.

-Test set score : 0.9228

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt

for C, marker in zip([0.001, 1, 100], ['o', '^', 'v']):
    lr_l1 = LogisticRegression(solver='liblinear', C=C, penalty="l1", max_iter=500).fit(X_train, y_train)
    print("C={:.3f} 인 l1 로지스틱 회귀의 훈련 정확도: {:.2f}".format(
          C, lr_l1.score(X_train, y_train)))
    print("C={:.3f} 인 l1 로지스틱 회귀의 테스트 정확도: {:.2f}".format(
          C, lr_l1.score(X_test, y_test)))
    plt.plot(lr_l1.coef_.T, marker, label="C={:.3f}".format(C))

#plt.xticks(range(cancer.data.shape[1]), cancer.feature_names, rotation=90)
#xlims = plt.xlim()
#plt.hlines(0, xlims[0], xlims[1])
#plt.xlim(xlims)
#plt.xlabel("특성")
#plt.ylabel("계수 크기")

plt.ylim(-5, 5)
plt.legend(loc=3)
plt.show() # 책에는 없음
```

```
C=0.001 인 l1 로지스틱 회귀의 훈련 정확도: 0.76
C=0.001 인 l1 로지스틱 회귀의 테스트 정확도: 0.76
C=1.000 인 l1 로지스틱 회귀의 훈련 정확도: 0.93
C=1.000 인 l1 로지스틱 회귀의 테스트 정확도: 0.92
C=100.000 인 l1 로지스틱 회귀의 훈련 정확도: 0.93
C=100.000 인 l1 로지스틱 회귀의 테스트 정확도: 0.92
```

○ LogisticRegression best model

- LogisticRegression model의 기본 매개변수를 통해 학습한 결과인 “Test set score = 0.9258”로 가장 높았다.

○ LogisticRegression analysis

- solver = ‘liblinear’와 L1규제를 적용했을 때, C = 0.001로 규제가 증가하여 가장 낮은 test set score=0.76이였다. 반대로 C값이 증가함에 따라 test set score=0.92로 수렴하는 듯 보였다.
- solver = ‘lbfgs’와 L2규제를 적용했을 때, C값에 관계없이 test set score=0.9228로 같았다.

2번

○ K-NN classifiers (n_neighbors = 2)일 때,

```
#  K-NN classifiers
# k=2

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors = 2).fit(X_train, y_train)

print("Training set score: {:.4f}".format(knn.score(X_train, y_train)))
print("Test set score: {:.4f}".format(knn.score(X_test, y_test)))
```

```
Training set score: 0.9857
Test set score: 0.9627
```

○ K-NN classifiers (n_neighbors = 3)일 때,

```
#  K-NN classifiers
# k=3

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors = 3).fit(X_train, y_train)

print("Training set score: {:.4f}".format(knn.score(X_train, y_train)))
print("Test set score: {:.4f}".format(knn.score(X_test, y_test)))
```

```
Training set score: 0.9867
Test set score: 0.9705
```

○ K-NN classifiers (n_neighbors = 4)일 때,

```
#  K-NN classifiers
# k=4

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors = 4).fit(X_train, y_train)

print("Training set score: {:.4f}".format(knn.score(X_train, y_train)))
print("Test set score: {:.4f}".format(knn.score(X_test, y_test)))
```

```
Training set score: 0.9830
Test set score: 0.9682
```

○ K-NN classifiers (n_neighbors = 5)일 때,

```
#  K-NN classifiers
# k=5

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors = 5).fit(X_train, y_train)

print("Training set score: {:.4f}".format(knn.score(X_train, y_train)))
print("Test set score: {:.4f}".format(knn.score(X_test, y_test)))
```

```
Training set score: 0.9819
Test set score: 0.9688
```

○ K-NN classifiers (n_neighbors = 6)일 때,

```
#  K-NN classifiers
# k=6

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors = 6).fit(X_train, y_train)

print("Training set score: {:.4f}".format(knn.score(X_train, y_train)))
print("Test set score: {:.4f}".format(knn.score(X_test, y_test)))
```

```
Training set score: 0.9797
Test set score: 0.9677
```

○ K-NN classifiers best model

-K-NN classifiers (n_neighbors = 3)일 때, “Test set score = 0.9705”로 가장 높았다.

○ SVM classifiers 기본 모델

```
# SVM classifiers
# kernel='rbf'

from sklearn.svm import SVC

SVM = SVC(kernel='rbf', C=1.0, gamma='auto').fit(X_train, y_train)

print("Training set score: {:.4f}".format(SVM.score(X_train, y_train)))
print("Test set score: {:.4f}".format(SVM.score(X_test, y_test)))
```

```
Training set score: 0.9430
Test set score: 0.9446
```

○ SVM classifiers 모델 비교(test set score)

	C=0.0001	C=0.001	C=0.01	C=0.1	C=1	C=2	C=5	C=10
‘RBF’	x	x	x	x	0.9446	0.9831	0.9561	x
‘linear’	0.9007	0.9302	0.9443	0.9472	0.9404	x	0.9338	0.9310
‘poly’	x	x	x	x	0.9771	0.7414	x	x

3번

○ SVM classifiers analysis

-gamma 매개변수는 가우시안 커널 폭의 역수에 해당한다.

-C는 선형 모델에서 사용한 것과 비슷한 규제 매개변수이다.

-SVM은 잘 작동하는 편이지만 매개변수 설정과 데이터 스케일에 매우 민감하다.(학습 가장 오래걸림)

-3가지 kernel에 C=1을 기반으로 높은 test set score을 detect 하였다. 그 중, kernel = ‘linear’가 정확도가 낮은 편이였는데, 이는 Mnist data set의 feature가 많기 때문에 선형으로 경계를 나눠 잘 적용되지 않다고 분석하였다.

-반면에, 곡선 형태로 경계를 나누는 형태인 ‘RBF’와 ‘poly’에서는 정확도가 높은 편이였다. 실패요인으로는 컴퓨터 계산량이 많아 gamma의 매개변수의 변화에 대해서는 파악하지 못한 점이다.

```
# SVM classifiers
# kernel='rbf'

from sklearn.svm import SVC

SVM = SVC(kernel='rbf', C=2).fit(X_train, y_train)

print("Training set score: {:.4f}".format(SVM.score(X_train, y_train)))
print("Test set score: {:.4f}".format(SVM.score(X_test, y_test)))
```

```
Training set score: 0.9957
Test set score: 0.9831
```

-결론적으로, kernel = ‘RBF’, C=2인 파라미터에서 test set score = 0.9831로 가장 높았다.

4번

○ Random forest classifiers 주요 코드

```
# Random forest classifiers

from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier(n_estimators=5, random_state=2) #n_estimators 생성할 트리 개수
forest.fit(X_train, y_train)

print("Training set score: {:.4f}".format(forest.score(X_train, y_train)))
print("Test set score: {:.4f}".format(forest.score(X_test, y_test)))
```

```
Training set score: 0.9938
Test set score: 0.9177
```

```
for n_estimators in [1, 5, 10, 20, 50, 100]:
    forest = RandomForestClassifier(n_estimators=n_estimators, random_state=2).fit(X_train, y_train)
    print("n_estimators={:.1f} 인 RandomForest의 훈련 정확도: {:.4f}".format(
          n_estimators, forest.score(X_train, y_train)))
    print("n_estimators={:.1f} 인 RandomForest의 테스트 정확도: {:.4f}".format(
          n_estimators, forest.score(X_test, y_test)))
```

```
n_estimators=1.0 인 RandomForest의 훈련 정확도: 0.9330
n_estimators=1.0 인 RandomForest의 테스트 정확도: 0.8251
n_estimators=5.0 인 RandomForest의 훈련 정확도: 0.9938
n_estimators=5.0 인 RandomForest의 테스트 정확도: 0.9177
n_estimators=10.0 인 RandomForest의 훈련 정확도: 0.9991
n_estimators=10.0 인 RandomForest의 테스트 정확도: 0.9481
n_estimators=20.0 인 RandomForest의 훈련 정확도: 0.9999
n_estimators=20.0 인 RandomForest의 테스트 정확도: 0.9616
n_estimators=50.0 인 RandomForest의 훈련 정확도: 1.0000
n_estimators=50.0 인 RandomForest의 테스트 정확도: 0.9679
n_estimators=100.0 인 RandomForest의 훈련 정확도: 1.0000
n_estimators=100.0 인 RandomForest의 테스트 정확도: 0.9698
```

```
n_estimators=1000.0 인 RandomForest의 훈련 정확도: 1.0000
n_estimators=1000.0 인 RandomForest의 테스트 정확도: 0.9711
```

○ Random forest classifiers 모델 비교(test set score)

n_estimators	1	5	10	20	50	100	1000	…
test score	0.8251	0.9177	0.9481	0.9616	0.9679	0.9698	0.9711	…

○ Random forest classifiers analysis

-n_estimators는 클수록 좋다. 더 많은 트리를 평균하면 과대적합을 줄여 더 안정적인 모델을 만든다.
-그러나 더 많은 트리는 더 많은 메모리와 긴 훈련 시간으로 이어진다.(제한적인 로컬 환경으로 인해 n_estimators = 1000까지만 학습하였음.)
-가용한 시간과 메모리만큼 많이 만드는 것이 좋다.
-사용하는 CPU 코어 개수에 비례해서 속도도 빨라진다.(코어 두 개 사용시, 훈련속도 두배 증가)
-매우 차원이 높고 희소한 데이터에는 잘 작동하지 않는다.
-랜덤 포레스트에서 성능을 좀 더 끌어내기 위한 트리로, gradient boosting 회귀 트리가 있다.

결과

○ Best results in the Table

	Logistic regression	K-NN	SVM	Random Forest
Accuracy	0.9258	0.9705	0.9831	0.9711

-결과론적으로, 4가지 classifier 모델 중 Mnist data set에 대해 가장 Accuracy가 높은 모델은 SVM이였다. 하지만 Random Forest의 경우 n_estimators = 1000까지만 학습하였기 때문에 Accuracy가 더 높아질 수 있다.
-MNIST 데이터 세트는 60,000개의 학습 데이터와 10,000개의 테스트 데이터로 구성되어 있다. 데이터 세트의 각 이미지는 가로, 세로가 각 28픽셀(pixel)로 28 x 28 = 784개의 숫자를 갖는 벡터로 생각할 수 있다.