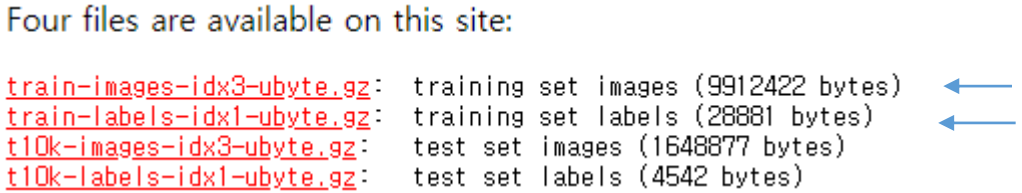
[Lab 5]

실습과제

- **이번 실습에서는 MNIST 데이터를 http://yann.lecun.com/exdb/mnist/ 에서 train-images와 train-labels 셋만 다운 받아서 사용합니다.**



- linear SVM과 nonlinear SVM의 성능을 비교 하시오. 성능 비교 방법은 각자 결정합니다.
- 다음의 내용을 보고서에 서술하시오.
- MNIST 데이터에 대한 설명: 특징값 개수와 타입, 클래스의 개수 등
- MNIST 데이터 파일의 포맷 (전처리를 하여 변형을 시켰다면 전처리 한 후의 포맷)
- 자신이 이용한 성능 비교 방법
- 성능 비교 결과

Sklearn 함수 소개

- 참고 자료

scikit-learn 의 설명은 https://scikit-learn.org/stable/index.html에서 제공됩니다.

선형 SVM 함수 : LinearSVC()

```
from sklearn.svm import LinearSVC
model = LinearSVC( )
clf = model.fit(X1, y1)     //training
clf.predict(X2)             //test
```

비선형 SVM 함수 : SVC( )

```
from sklearn.svm import SVC
model = SVC(kernel='rbf', C=1, gamma=0.1)
//rbf kernel k(x1,x2)=exp(-gamma||x1-x2||^2)
clf = model.fit(X1, y1)     //training
clf.predict(X2)             //test
```

Sklearn train_test_split()

- train_test_split()

```
from sklearn.model_selection import train_test_split
train_test_split(arrays, test_size, train_size, random_state, shuffle, stratify)
```

- Parameter

- **arrays** : 분할시킬 데이터를 입력 *(Python list, Numpy array, Pandas dataframe 등..)*
- **test_size** : 테스트 데이터셋의 비율(float)이나 개수(int) *(default = 0.25)*
- **train_size** : 학습 데이터셋의 비율(float)이나 개수(int) *(default = test_size의 나머지)*
- **random_state** : 데이터 분할 시 셔플이 이루어지는데 이를 위한 시드 값 *(int나 RandomState로 입력)*
- **shuffle** : 셔플 여부 설정 *(default = True)*
- **stratify** : 지정한 Data의 비율을 유지한다. 예를 들어, Label Set인 Y가 25%의 0과 75%의 1로 이루어진 Binary Set일 때, stratify=Y로 설정하면 나누어진 데이터셋들도 0과 1을 각각 25%, 75%로 유지한 채 분할된다.

- Return

- **X_train, X_test, Y_train, Y_test** : arrays에 데이터와 레이블을 둘 다 넣었을 경우의 반환이며, 데이터와 레이블의 순서쌍은 유지됩니다.
- **X_train, X_test** : arrays에 레이블 없이 데이터만 넣었을 경우의 반환됩니다.

Sklearn train_test_split()

- train_test_split()

- Example

```
import numpy as np
from sklearn.model_selection import train_test_split
X, y = np.arange(10).reshape((5, 2)), range(5)
list(y)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=42)
print(X_train)
# [[4 5]
#  [0 1]
#  [6 7]]
print(y_train)
# [2, 0, 3]
print(X_test)
# [[2 3]
#  [8 9]]
print(y_test)
# [1, 4]


# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
```

Sklearn Cross_val_score()

- cross_val_score()

```
from sklearn.model_selection import cross_val_score
cross_val_score(estimator, X, y=None, scoring=None, cv=None, n_jobs=1, verbose=0, fit_params=None,
pre_dispatch='2*n_jobs' )
```

- Parameter

- **estimator** : 학습을 할 모델 (ex => estimator=knn)
- **X** : 학습시킬 훈련 데이터 세트
- **y** : 학습시킬 훈련 데이터 세트의 Label*(default = None)*
- **scoring** : 각 모델에서 사용할 평가 방법입니다. *(string, callable or None, optional, default: None)*
- **cv** : int 또는 kfold로 fold의 수를 의미합니다. *(default = None)*

- Return

- **scores**: 교차 검증 각 실행에 대한 점수의 배열을 반환합니다.

Sklearn Cross_val_score()

- cross_val_score()

- Example()

```
from sklearn import datasets, linear_model
from sklearn.model_selection import cross_val_score

diabetes = datasets.load_diabetes()
X = diabetes.data[:150]
y = diabetes.target[:150]
lasso = linear_model.Lasso()
cross_val_score(lasso, X, y, cv=3)
#[0.33150734 0.08022311 0.03531764]

#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html
```

Sklearn을 활용한 KNN

- Iris 데이터를 활용한 KNN

```
import numpy as np

data = np.loadtxt("iris.csv", delimiter=",")
iris_data = data[:, 0:4]
iris_label = data[:, 4:5]


# 테스트 세트와 트레이닝 세트 분리하기
from sklearn.model_selection import train_test_split
training_data, validation_data, training_labels, validation_labels = train_test_split(iris_data, iris_label,
                                                                                      test_size=0.3, random_state=0)
```

Sklearn을 활용한 KNN

- Iris 데이터를 활용한 KNN

```
# KNN
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=1)

classifier.fit(training_data, training_labels.ravel())
print(classifier.score(validation_data, validation_labels))
#0.9777777777777777

from sklearn.model_selection import cross_val_score
# 교차 검증 수행
scores = cross_val_score(classifier, training_data, training_labels.ravel(), scoring='accuracy')
print(scores)
#[0.85714286 1.         1.         0.9047619  0.95238095]
print(np.mean(scores))
#0.9428571428571428
```