대규모 데이터 학습

대규모 데이터(big data)의 경우에는 메모리 등의 문제로 특정한 모형은 사용할 수 없는 경우가 많다. 이 때는

- 사전 확률분포를 설정할 수 있는 생성 모형
- 시작 가중치를 설정할 수 있는 모형

등을 이용하고 전체 데이터를 처리 가능한 작은 주각으로 나누어 학습을 시키는 점진적 학습 방법을 사용한다.

In [1]:

```
from sklearn.datasets import fetch_covtype
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

covtype = fetch_covtype(shuffle=True, random_state=0)
X_covtype = covtype.data
y_covtype = covtype.target - 1
classes = np.unique(y_covtype)
X_train, X_test, y_train, y_test = train_test_split(X_covtype, y_covtype)

scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

def read_Xy(start, end):
    # 실무에서는 파일이나 데이터베이스에서 읽어온다.
    idx = list(range(start, min(len(y_train) - 1, end)))
    X = X_train[idx, :]
    y = y_train[idx]
    return X, y
```

SGD

퍼셉트론 모형은 가중치를 계속 업데이트하므로 일부 데이터를 사용하여 구한 가중치를 다음 단계에서 초기 가중치로 사용할 수 있다.

In [2]:

```
%%time

from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

model = SGDClassifier(random_state=0)
n_split = 10
n_X = len(y_train) // n_split
n_epoch = 10
for epoch in range(n_epoch):
    for n in range(n_split):
        X, y = read_Xy(n * n_X, (n + 1) * n_X)
        model.partial_fit(X, y, classes=classes)
    accuracy_train = accuracy_score(y_train, model.predict(X_train))
    accuracy_test = accuracy_score(y_test, model.predict(X_test))
    print("epoch={:d} train acc={:5.3f} test acc={:5.3f}".format(epoch, accuracy_train, accuracy_tes
```

```
epoch=0 train acc=0.707 test acc=0.707
epoch=1 train acc=0.710 test acc=0.710
epoch=2 train acc=0.711 test acc=0.710
epoch=3 train acc=0.711 test acc=0.711
epoch=4 train acc=0.711 test acc=0.711
epoch=5 train acc=0.711 test acc=0.711
epoch=6 train acc=0.711 test acc=0.711
epoch=7 train acc=0.710 test acc=0.710
epoch=8 train acc=0.711 test acc=0.711
epoch=9 train acc=0.711 test acc=0.711
CPU times: user 15.2 s, sys: 330 ms, total: 15.6 s
Wall time: 9.21 s
```

나이브베이즈 모형

나이브베이즈 모형과 같은 생성모형은 일부 데이터를 이용하여 구한 확률분포를 사전확률분포로 사용할 수 있다.

In [3]:

```
%%time

from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

model = BernoulliNB(alpha=0.1)

n_split = 10
n_X = len(y_train) // n_split
for n in range(n_split):
    X, y = read_Xy(n * n_X, (n + 1) * n_X)
    model.partial_fit(X, y, classes=classes)
    accuracy_train = accuracy_score(y_train, model.predict(X_train))
    accuracy_test = accuracy_score(y_test, model.predict(X_test))
    print("n={:d} train accuracy={:5.3f} test accuracy={:5.3f}".format(n, accuracy_train, accuracy_t
```

```
n=0 train accuracy=0.630 test accuracy=0.628
n=1 train accuracy=0.630 test accuracy=0.629
n=2 train accuracy=0.632 test accuracy=0.630
n=3 train accuracy=0.633 test accuracy=0.632
n=4 train accuracy=0.633 test accuracy=0.631
n=5 train accuracy=0.632 test accuracy=0.631
n=6 train accuracy=0.632 test accuracy=0.631
n=7 train accuracy=0.632 test accuracy=0.630
n=8 train accuracy=0.632 test accuracy=0.630
n=9 train accuracy=0.632 test accuracy=0.630
CPU times: user 10.1 s, sys: 920 ms, total: 11 s
Wall time: 2.78 s
```

그레디언트 부스팅

그레디언트 부스팅에서는 초기 커미티 멤버로 일부 데이터를 사용하여 학습한 모형을 사용할 수 있다.

In [4]:

```
%%time

from lightgbm import train, Dataset
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

params = {
    'objective': 'multiclass',
    "num_class": len(classes),
    'learning_rate': 0.2,
    'seed': 0,
}

n_split = 10
n_X = len(y_train) // n_split
num_tree = 10
model = None
for n in range(n_split):
    X, y = read_Xy(n * n_X, (n + 1) * n_X)
    model = train(params, init_model=model, train_set=Dataset(X, y),
                  keep_training_booster=False, num_boost_round=num_tree)
    accuracy_train = accuracy_score(y_train, np.argmax(model.predict(X_train), axis=1))
    accuracy_test = accuracy_score(y_test, np.argmax(model.predict(X_test), axis=1))
    print("n={:d} train accuracy={:5.3f} test accuracy={:5.3f}".format(n, accuracy_train, accuracy_t
```

```
n=0 train accuracy=0.769 test accuracy=0.767
n=1 train accuracy=0.792 test accuracy=0.788
n=2 train accuracy=0.806 test accuracy=0.802
n=3 train accuracy=0.816 test accuracy=0.812
n=4 train accuracy=0.812 test accuracy=0.807
n=5 train accuracy=0.808 test accuracy=0.804
n=6 train accuracy=0.805 test accuracy=0.800
n=7 train accuracy=0.793 test accuracy=0.788
n=8 train accuracy=0.795 test accuracy=0.790
n=9 train accuracy=0.794 test accuracy=0.789
CPU times: user 3min 17s, sys: 3.59 s, total: 3min 20s
Wall time: 1min 3s
```

Random Forest

랜덤 포레스트와 같은 앙상블 모형에서는 일부 데이터를 사용한 모형을 개별 분류기로 사용할 수 있다.

In [5]:

```
%%time

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

n_split = 10
n_X = len(y_train) // n_split
num_tree_ini = 10
num_tree_step = 10
model = RandomForestClassifier(n_estimators=num_tree_ini, warm_start=True)
for n in range(n_split):
    X, y = read_Xy(n * n_X, (n + 1) * n_X)
    model.fit(X, y)
    accuracy_train = accuracy_score(y_train, model.predict(X_train))
    accuracy_test = accuracy_score(y_test, model.predict(X_test))
    print("epoch={:d} train accuracy={:5.3f} test accuracy={:5.3f}".format(n, accuracy_train, accura

    model.n_estimators += num_tree_step
```

```
epoch=0 train accuracy=0.868 test accuracy=0.855
epoch=1 train accuracy=0.892 test accuracy=0.874
epoch=2 train accuracy=0.898 test accuracy=0.880
epoch=3 train accuracy=0.902 test accuracy=0.884
epoch=4 train accuracy=0.904 test accuracy=0.885
epoch=5 train accuracy=0.906 test accuracy=0.886
epoch=6 train accuracy=0.906 test accuracy=0.887
epoch=7 train accuracy=0.907 test accuracy=0.887
epoch=8 train accuracy=0.907 test accuracy=0.888
epoch=9 train accuracy=0.907 test accuracy=0.888
CPU times: user 4min 46s, sys: 11.4 s, total: 4min 57s
Wall time: 1min 42s
```