資料科學, Fall 2017

Homework #2 Spam Letter Classification

---

# Model Design

For SVM, if we pick C > 1, for instance, C = 100 or 1000, it seems to be overfitting, that is, for training set it will be 99% or more. therefore we picked C = 1.
For Others, the original set of parameters are good enough, so I just modified a little bit, please see the below section "Modified Design".

# Model Validation (5-fold)

if test_size = 0.13 (train_size = 0.87), that is, train_X shape is (3999, 58)
the below test_y are given since that I split the spambase.csv myself for these experiments.

1. R (Logistic Regression)

**model.score(train_X, train_y):**
0.94023230291
**Cross Validation(5-fold, negative mean square error):**
0.07 (+/- 0.02)
**model.score(test_X, test_y):**
0.92857109283
**Cross Validation(5-fold, negative mean square error):**
0.10 (+/- 0.06)

2. D (Decision Tree)

**model.score(train_X, train_y):**
0.99924981245s
**Cross Validation(5-fold, negative mean square error):**
0.09 (+/- 0.02)
**model.score(test_X, test_y):**
0.93023255814
**Cross Validation(5-fold, negative mean square error):**
0.12 (+/- 0.06)

3. S (SVM)

**model.score(train_X, train_y):**
0.947736934234
**Cross Validation(5-fold, negative mean square error):**
0.07 (+/- 0.02)
**model.score(test_X, test_y):**
0.941860465116
**Cross Validation(5-fold, negative mean square error):**

0.08 (+/- 0.04)

4. N (Neural Network)

**model.score(train_X, train_y):**
0.975743935984
**Cross Validation(5-fold, negative mean square error):**
0.06 (+/- 0.01)
**model.score(test_X, test_y):**
0.93853820598
**Cross Validation(5-fold, negative mean square error):**
0.08 (+/- 0.03)

## Some modified design

```
def lrModel(train_X, train_y):

    lr = linear_model.LogisticRegression(
        penalty='l2', solver='liblinear', multi_class='ovr', verbose=0,
n_jobs=1)
    lr.fit(train_X, train_y)
    return lr

def dctModel(train_X, train_y):

    dct = sklearn.tree.DecisionTreeClassifier()
    dct.fit(train_X, train_y)
    return dct

def svcModel(train_X, train_y):

    svc = sklearn.svm.SVC(
        C=1,
        kernel='rbf',
        degree=3,
        gamma='auto',
        decision_function_shape='ovr')

    svc.fit(train_X, train_y)

    return svc

def mlpModel(train_X, train_y):
    mlp = MLPClassifier(hidden_layer_sizes = (15,), max_iter=2000)
```

```
    mlp.fit(train_X, train_y)
    return mlp
```

## Preprocessing

In spite of the design described above, I also did some preprocessing as shown below:

```
scaler = preprocessing.StandardScaler().fit(train_X)
    train_X = scaler.transform(train_X)
    test_X = scaler.transform(test_X)
```