309551101 郭育麟
Pattern Recognition

# *Part 1*

1.
(10%) K-fold data partition: Implement the K-fold cross-validation function. Your function should take K as an argument and return a list of lists (len(list) should equal to K), which contains K elements. Each element is a list contains two parts, the first part contains the index of all training folds (index_x_train, index_y_train), e.g. Fold 2 to Fold 5 in split 1. The second part contains the index of validation fold, e.g. Fold 1 in split 1 (index_x_val, index_y_val)
Note: You need to handle if the sample size is not divisible by K. Using the strategy from sklearn. The first n_samples % n_splits folds have size n_samples // n_splits + 1, other folds have size n_samples // n_splits, where n_samples is the number of samples, n_splits is K, % stands for modulus, // stands for integer division. See this post for more details
Note: Each of the samples should be used exactly once as the validation data
Note: Please shuffle your data before partition

K is the parameter and represent that has K choice for training set and validation set . Due to need to handle the sample size is not divisible by K , I first create an array to represent the number of each partition should choose , then use “random” to choose index k times . Then have k partition , so I could choose one partition for validation set and another k-1 partition for training set , and each partition would be once the validation set .

```python
def cross_validation(x_train, y_train, k=5):
    # Random is equivalent to shufle
    number_all = x_train.shape[0]
    number_plus = number_all % k
    number_choose = [int(number_all/k) for i in range(k)]
    for i in range(number_plus):
        number_choose[i] += 1
    all_ = [i for i in range(0, number_all)]
    k_partition_index = []
    for number in number_choose:
        choose = random.sample(all_, number)
        k_partition_index.append([choose])
        all_ = list(set(all_) - set(choose))

    all_ = [i for i in range(0, number_all)]
    kfold_data = []
    for i in range(k):
        val_index = k_partition_index[i][0]
        train_index = np.array(list(set(all_) - set(val_index)))
        val_index = np.array(val_index)
        kfold_data.append([train_index, val_index])
    return kfold_data
```

2.

(30%) Grid Search & Cross-validation: using sklearn.svm.SVC to train a classifier on the provided train set and conduct the grid search of “C” and “gamma”, “kernel’=’rbf’ to find the best hyperparameters by cross-validation. Print the best hyperparameters you found.

Note: We suggest use K=5

In this step , let “C” and “Gamma” has multiple choice , which C = [0.01, 0.1, 1.0, 10.0, 100.0, 1000.0, 10000.0] , Gamma = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000] and use grid search to find the best model parameters .

```python
def Training(x_train, y_train, clf, kfold_data):
    times = len(kfold_data)
    total_accuracy = 0
    for i in range(times):
        training_index = kfold_data[i][0]
        val_index = kfold_data[i][1]
        training_data = x_train[training_index, :]
        training_label = y_train[training_index]
        val_data = x_train[val_index, :]
        val_label = y_train[val_index]
        clf.fit(training_data, training_label)
        accuracy = clf.score(val_data, val_label)
        total_accuracy += accuracy
    return total_accuracy / times
```

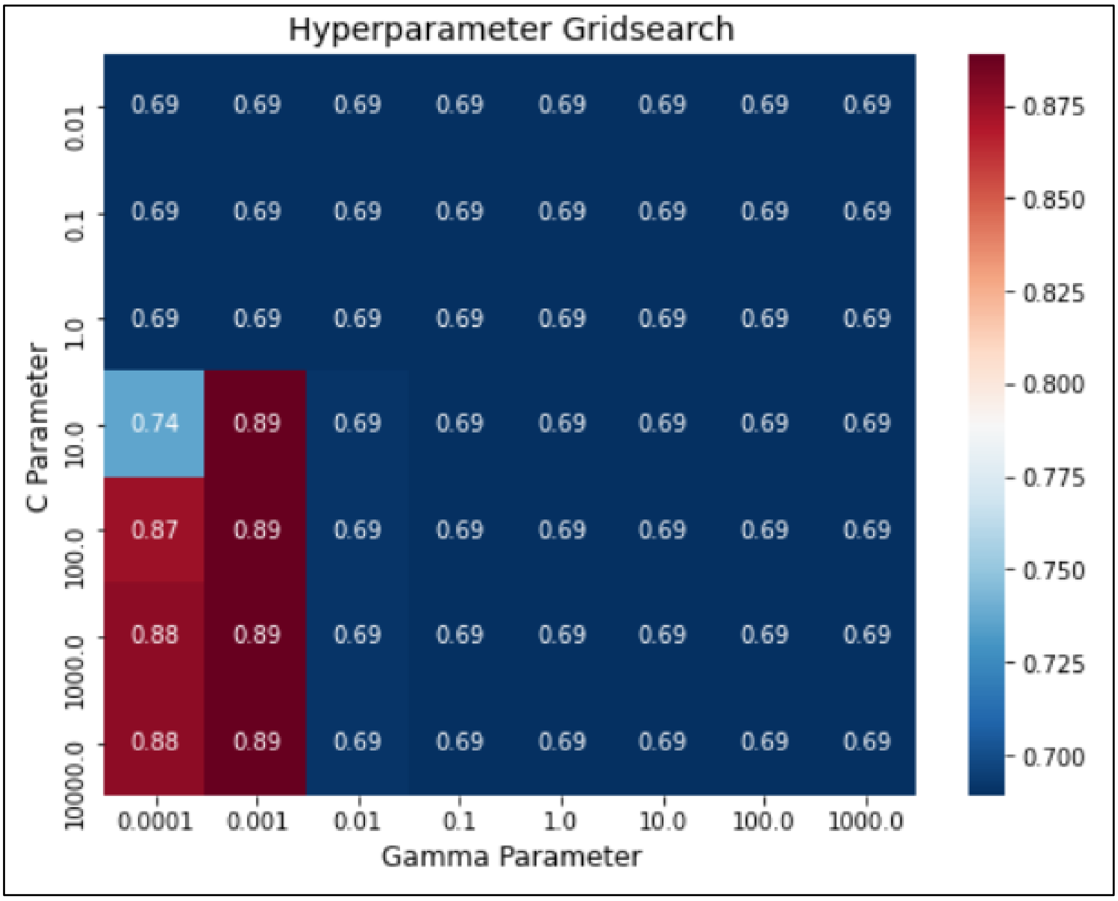
```python
def Grid_Search(C_list, gamma_list, x_train, y_train, kfold_data, method):
    best_accuracy = 0
    best_C = 0
    best_gamma = 0
    best_model = None
    gird_array = np.zeros((len(C_list), len(gamma_list) ))
    for i, gamma in enumerate(gamma_list):
        for j, C in enumerate(C_list):
            if method == "SVC" :
                clf = SVC(C = C, kernel='rbf', gamma = gamma)
            else :
                clf = SVR(C = C, kernel='rbf', gamma = gamma, epsilon = 0.1)
            accuracy = Training(x_train, y_train, clf, kfold_data)
            gird_array[j, i] = accuracy
            if accuracy > best_accuracy :
                best_accuracy = accuracy
                best_C = C
                best_gamma = gamma
                best_model = clf
    return best_accuracy, best_C, best_gamma, gird_array, best_model
```

3.

(10%) Plot the grid search results of your SVM. The x, y represent the hyperparameters of “gamma” and “C”, respectively. And the color represents the average score of validation folds.

```python
import seaborn as sn
import matplotlib.pyplot as plt

df = pd.DataFrame(gird_array, index = [i for i in C_list],
                  columns = [i for i in gamma_list])
plt.figure(figsize = (8,6))
plt.title("Hyperparameter Gridsearch", fontsize = "14")
sn.heatmap(df, annot=True, cmap='RdBu_r')
plt.xlabel("Gamma Parameter", fontsize = "12")
plt.ylabel("C Parameter", fontsize = "12")
plt.show()
```




4.

(15%) Train your SVM model by the best hyperparameters you found from question 2 on the whole training set and evaluate the performance on the test set.

Note: Your accuracy scores should be higher than 0.85

```python
y_pred = best_model.predict(x_test)
print(f"Accuracy score: {accuracy_score(y_pred, y_test):.04f}")
```

```
Accuracy score: 0.9115
```

5.

(15%) Consider the dataset used in HW1 for regression. Please redo the above questions 2 ~ 4 with the dataset replaced by that used in HW1, while the task is changed from classification to regression. You should use the SVM regression model RBF kernel with grid search for hyperparameters and K-fold cross-validation (you can use any K for cross-validation). Then compare the linear regression model you have implemented in HW1 with SVM by showing the Mean Square Errors of both models on the test set.

In [13]:
```
train_df = pd.read_csv("../HW1/train_data.csv")
x_train = train_df['x_train'].to_numpy().reshape(-1,1)
y_train = train_df['y_train'].to_numpy().reshape(-1,1)

test_df = pd.read_csv("../HW1/test_data.csv")
x_test = test_df['x_test'].to_numpy().reshape(-1,1)
y_test = test_df['y_test'].to_numpy().reshape(-1,1)

#from HW1
w0_MSE = -0.0012762766165574284
w1_MSE = 0.452720248401306
y_pred_MSE = w0_MSE + w1_MSE*x_test
MSE_loss = np.sum((y_test-y_pred_MSE)**2) / len(x_test)

y_train = train_df['y_train'].to_numpy().reshape(-1)
y_test = test_df['y_test'].to_numpy().reshape(-1)

method = "SVR"
kfold_data = cross_validation(x_train, y_train, k=5)
C_list = [0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]
gamma_list = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100]
best_accuracy, best_C, best_gamma, gird_array, best_model = Grid_Search(C_list, gamma_list, x_train, y_train, kfold_data, me
y_hat = best_model.predict(x_test)
SVM_loss = np.sum((y_test-y_hat)**2) / len(x_test)
```

In [14]:
```
print(f"Square error of Linear regression: {MSE_loss}")
print(f"Square error of SVM regresssion model: {SVM_loss}")
```

```
Square error of Linear regression: 0.49090350083705264
Square error of SVM regresssion model: 0.4959807581738682
```

# Part2

1.

Given a valid kernel $k_1(x, x')$, prove that 1) $k(x, x') = ck_1(x, x')$ and 2) $k(x, x') = f(x)\, k_1(x, x')\, f(x')$ are valid kernels, where $c > 0$ is a positive constant and $f(\cdot)$ is any real-valued function.

$k_1(x, x')$ is a valid kernel

$$\Leftrightarrow K' = \begin{bmatrix} k_1(x_1, x_1) & \dots & k_1(x_1, x_n) \\ \vdots & \ddots & \vdots \\ k_1(x_n, x_1) & \dots & k_1(x_n, x_n) \end{bmatrix} \text{ is positive semidefinite}$$

$$\Leftrightarrow \ \forall\, a \in R^n \ a^T K' a \geq 0$$

(1)

$K(x, x') = c\, k_1(x, x')$

$$a^T K a = [a_1, a_2, \dots, a_n] \begin{bmatrix} ck_1(x_1, x_1) & \dots & ck_1(x_1, x_n) \\ \vdots & \ddots & \vdots \\ ck_1(x_n, x_1) & \dots & ck_1(x_n, x_n) \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_1 \end{bmatrix} = c * a^T K' a \geq 0$$

$\therefore$ K is a valid kernel function

(2)

$\mathrm{K}(\mathrm{x}, \mathrm{x}') = f(x) k_1(x, x') f(x')$

$$\mathrm{a}^{\mathrm{T}} K a = [a_1 \ , a_2 \ \ldots a_n] \begin{bmatrix} f(x_1) k_1(x_1, x_1) f(x_1) & \cdots & f(x_1) k_1(x_1, x_n) f(x_n) \\ \vdots & \ddots & \vdots \\ f(x_n) k_1(x_n, x_1) f(x_1) & \cdots & f(x_n) k_1(x_n, x_n) f(x_n) \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix}$$

$$= [\mathrm{a}_1 f(x_1) \ldots a_n f(x_n)] K' \begin{bmatrix} a_1 f(x_1) \\ \vdots \\ a_n f(x_n) \end{bmatrix} \geq 0$$ $\therefore$ K is a valid kernel function