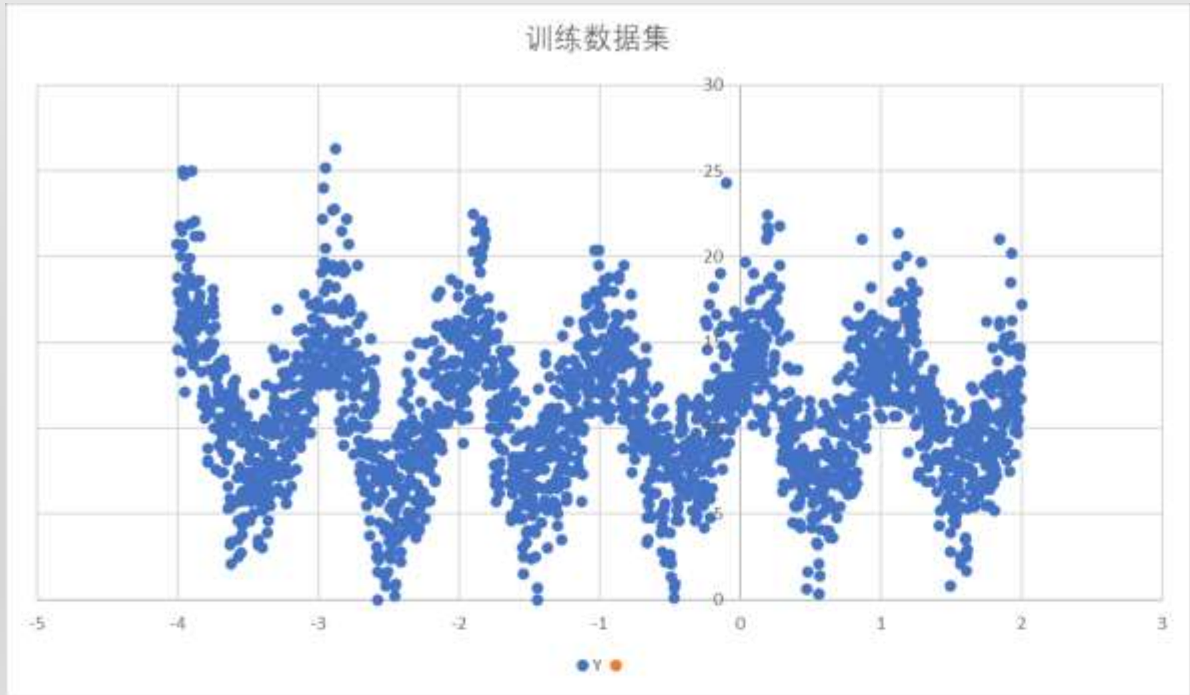
实验2. Kernel Method

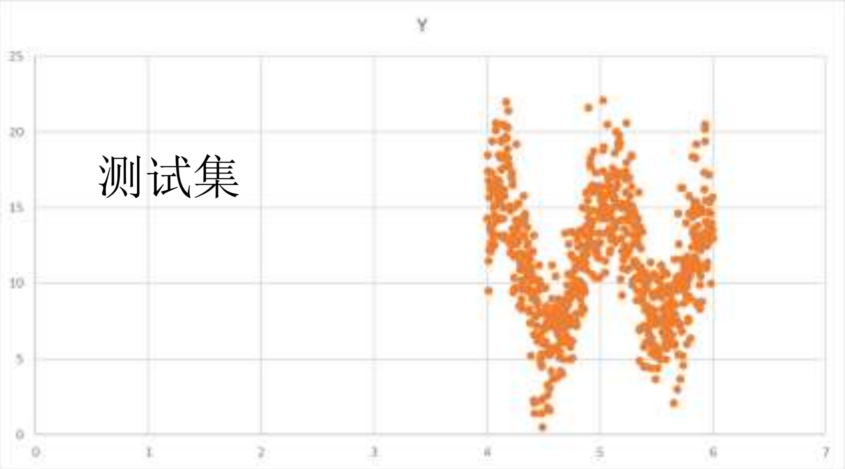
Lab 2.1 Kernel Ridge Regression

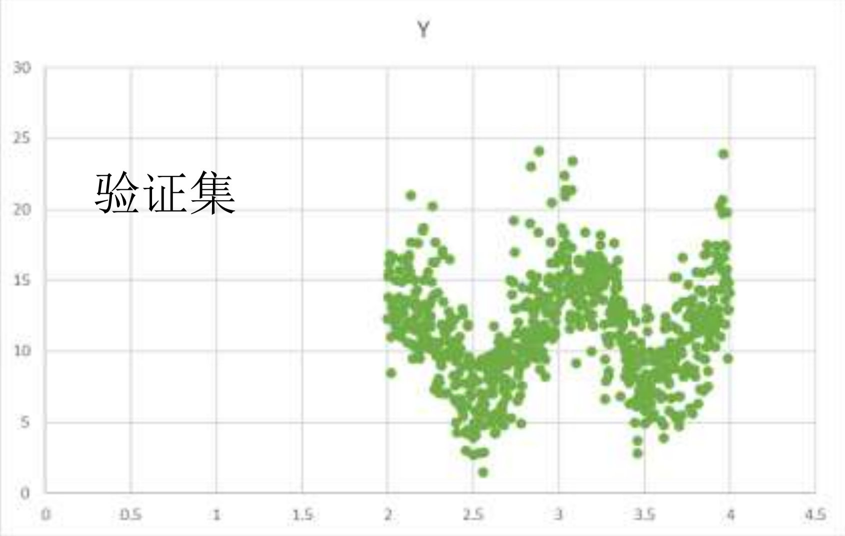
datasets

任务1. 读取数据集并可视化。

给定三个数据集：data_train.csv ($N = 2400$)，data_valid.csv (验证集)，data_test.csv（测试集），读入三个数据集，并可视化.







Kernel function

任务2. 实现三个核函数，并通过函数图像分析核函数性质

- **Linear kernel** (linear_kernel) **:** $k(\mathbf{x},\mathbf{z}) = \sum_{f=1}^{F} x_f z_f = \mathbf{x}^\top \mathbf{z}$
- **polynomial kernel** (poly_kernel) **:** $k(\mathbf{x},\mathbf{z}) = (\mathbf{x}^\top \mathbf{z} + c)^p$
- **Gaussian kernel** (sqexp_kernel) **:** $k(\mathbf{x},\mathbf{z}) = \exp\left(-\frac{(\mathbf{x}-\mathbf{z})^\top(\mathbf{x}-\mathbf{z})}{\ell^2}\right) = \exp\left(-\frac{\sum_{f=1}^{F}\left(x_f - z_f\right)^2}{\ell^2}\right)$
- **Periodic kernel** (periodic_kernel) **:** $k(\mathbf{x},\mathbf{z}) = \exp\left(-\frac{1}{2}\frac{\left(\sin\left(\frac{\pi}{p}(\mathbf{x}-\mathbf{z})\right)\right)^2}{\ell^2}\right)$

2.1 基于python和numpy实现4个核函数（kernel function）.

2.2 对于上述4个核函数，分别绘制出两个函数: $k(x,1)$， $k(x,0)$的图像

2.3 对于poly_kernel, GK和PeriodicK，通过绘制不同超参数的$k(x,1)$， $k(x,0)$ （ $x \in [-6,6]$ ）的图像，并分析超参数ℓ， p对于两个核函数的影响，得出具体结论

Kernel function

```
def  linear_kernel(x_QF, x_train_NF=None):
    ''' Evaluate linear kernel matrix between two datasets.

    Will compute the kernel function for all possible pairs of feature vectors,
    one from the query dataset, one from the reference training dataset.

                Args
                ----
    x_QF : 2D numpy array, shape (Q, F) = (n_query_examples, n_features)
                         Feature array for *query* dataset
                         Each row corresponds to the feature vector on example

    x_train_NF : 2D numpy array, shape (N, F) = (n_train_examples, n_features)
                         Feature array for reference *training* dataset
                         Each row corresponds to the feature vector on example

    Returns
    -------
    k_QN : 2D numpy array, shape (Q, N)
        Entry at index (q,n) corresponds to the kernel function evaluated
        at the feature vectors x_QF[q] and x_train_NF[n]
    '''
```

```
>>> np.set_printoptions(precision=3, suppress=1)

# Kernel evaluations with F=1 features
>>> x_zero_11 = np.asarray([[0.0]])
>>> x_one_11 = np.asarray([[1.0]])

# Linear kernel k(0,0) should be zero
>>> k_11 = calc_linear_kernel(x_zero_11, x_zero_11)
>>> k_11.ndim
2
>>> k_11
array([[0.]])

# Linear kernel k(1,1) should be one
>>> calc_linear_kernel(x_one_11, x_one_11)
array([[1.]])

# Linear kernel k(1, 3.456) should be 3.456
>>> calc_linear_kernel(x_one_11, 3.456 * x_one_11)
array([[3.456]])


# Part 2: Kernel evaluations with F=2 features and several examples at once
>>> x_train_32 = np.asarray([[0.0, 0.0], [1.0, 1.0], [2.0, 2.0]])
>>> calc_linear_kernel(x_train_32, x_train_32)
array([[0., 0., 0.],
       [0., 2., 4.],
       [0., 4., 8.]])
```

Kernel function

```
def sqexp_kernel(x_QF, x_train_NF=None, length_scale=1.0):
    ''' Evaluate squared-exponential kernel matrix between two datasets.
    Will compute the kernel function for all possible pairs of feature vectors,
    one from the query dataset, one from the reference training dataset.
    Args
            ----
    x_QF : 2D numpy array, shape (Q, F) = (n_query_examples, n_features)
                  Feature array for *query* dataset
                  Each row corresponds to the feature vector on example

    x_train_NF : 2D numpy array, shape (N, F) = (n_train_examples, n_features)
                  Feature array for reference *training* dataset
                  Each row corresponds to the feature vector on example

    Returns
    --------
    k_QN : 2D numpy array, shape (Q, N)
        Entry at index (q,n) corresponds to the kernel function evaluated
        at the feature vectors x_QF[q] and x_train_NF[n]
    '''
```

```
>>> np.set_printoptions(precision=3, suppress=1)

# Example 1: Simple kernel evaluations with F=1 features
>>> x_zero_11 = np.asarray([[0.0]])
>>> x_one_11 = np.asarray([[1.0]])
>>> k_11 = sqexp_kernel(x_zero_11, x_zero_11, length_scale=1.0)
>>> k_11.ndim
2
>>> k_11
array([[1.]])
>>> sqexp_kernel(x_one_11, x_one_11, length_scale=1.0)
array([[1.]])
>>> sqexp_kernel(x_one_11, x_zero_11, length_scale=1.0)
array([[0.368]])


# Example 2: Kernel evaluations with F=2 features and several examples at once
>>> x_train_32 = np.asarray([[0.0, 0.0], [1.0, 1.0], [2.0, 2.0]])
>>> k_33 = sqexp_kernel(x_train_32, x_train_32)
>>> k_33
array([[1.   , 0.135, 0.   ],
       [0.135, 1.   , 0.135],
       [0.   , 0.135, 1.   ]])
```

Kernel function

```
def periodic_kernel(x_QF, x_train_NF=None, length_scale=1.0, period=1.0):
    ''' Evaluate periodic kernel to produce matrix between two datasets.

    Will compute the kernel function for all possible pairs of feature vectors,
    one from the query dataset, one from the reference training dataset.

    Args
    ----
    x_QF : 2D numpy array, shape (Q, F) = (n_query_examples, n_features)
        Feature array for *query* dataset
        Each row corresponds to the feature vector on example

    x_train_NF : 2D numpy array, shape (N, F) = (n_train_examples, n_features)
        Feature array for reference *training* dataset
        Each row corresponds to the feature vector on example

    Returns
    -------
    k_QN : 2D numpy array, shape (Q, N)
        Entry at index (q,n) corresponds to the kernel function evaluated
        at the feature vectors x_QF[q] and x_train_NF[n]
    '''
```

```
Examples
--------
>>> np.set_printoptions(precision=3, suppress=1)

# Part 1: Simple kernel evaluations with F=1 features
>>> x_zero_11 = np.asarray([[0.0]])
>>> x_one_11 = np.asarray([[1.0]])

# Kernel of x=0.0 with itself should be 1.0
>>> k_11 = periodic_kernel(x_zero_11, x_zero_11, length_scale=2.0, period=0.3)
>>> k_11.ndim
2
>>> k_11
array([[1.]])

# Kernel of x and z=x+period should be 1.0
>>> p = 0.3
>>> periodic_kernel(x_one_11, x_one_11 + p, length_scale=2.0, period=p)
array([[1.]])
>>> periodic_kernel(x_one_11, x_one_11 + 3 * p, length_scale=2.0, period=p)
array([[1.]])

# Part 2: Kernel evaluations with several examples at once (still F=1)
>>> x_train_31 = np.asarray([[0.0], [1.0], [2.0]])
>>> periodic_kernel(x_train_31, x_train_31, length_scale=2.0, period=0.95)
array([[1.   , 0.997, 0.987],
       [0.997, 1.   , 0.997],
       [0.987, 0.997, 1.   ]])
>>> x_test_21 = np.asarray([[-0.5], [0.5]])
>>> periodic_kernel(x_test_21, x_train_31, length_scale=2.0, period=0.95)
array([[0.883, 0.889, 0.9  ],
       [0.883, 0.883, 0.889]])
```

kernel Ridge Regression

$$\hat{\mathbf{y}} = \mathbf{K}_{\hat{\mathbf{X}}\mathbf{X}}(\sigma^2\lambda\mathbf{I} + \mathbf{K}_{\mathbf{X}\mathbf{X}})^{-1}\mathbf{y}$$

任务3. 实现kernel Ridge Regression（KRR）模型，该模型能够嵌入用户指定的任何核函数。基于前述4个核函数，分别实现三个不同的KRR模型，并用“data_train.csv”**进行模型训练；基于**“data_valid.csv”**用grid search（Scikit-Learn GridSearchCV）方法选择核函数中最优超参数；最后，在**data_test.csv上进行模型验证，给出不同核回归模型的RMSE对比结果，并对该结果进行分析。

在同一幅图上绘制“训练集”“测试集”，并绘制三个不同核回归模型的“回归曲线”

Lab 2.2 Kernel SVM for Classification

datasets

任务1. 在二维平面上分别以 $X_1 \sim N\left([-3,-3],\begin{bmatrix}2,-1\\-1,2\end{bmatrix}\right)$, $X_2 \sim N\left([3,3],\begin{bmatrix}2,-1\\-1,2\end{bmatrix}\right)$ 各随机生成80个样本点（ X_1 样本点作为-1类， X_2 样本点作为+1类），按照40:20:20划分为“训练集”、“验证集”、“测试集”，并绘制。

任务2. 基于python和numpy，以及cvxopt（cvxopt.org ）设计并实现核SVM模型 (需自己实现，不允许直接调用已有库中SVM的实现). 分别利用linear_kernel 、 ploy_kernel 、 sqexp_kernel 、periodic_kernel实现四个不同的核SVM模型。并用“训练集”进行模型训练；基于“验证集”用grid search（Scikit-Learn GridSearchCV）方法选择核函数中最优超参数；最后，在测试集上进行模型验证，给出不同核回归模型的AUC分类对比结果，并对该结果进行分析。**对于不同的核SVM模型，绘制“训练集” “测试集”，并绘制分类曲线、间隔曲线以及标识出支持向量**