SVM

1. 算法原理

1）介绍

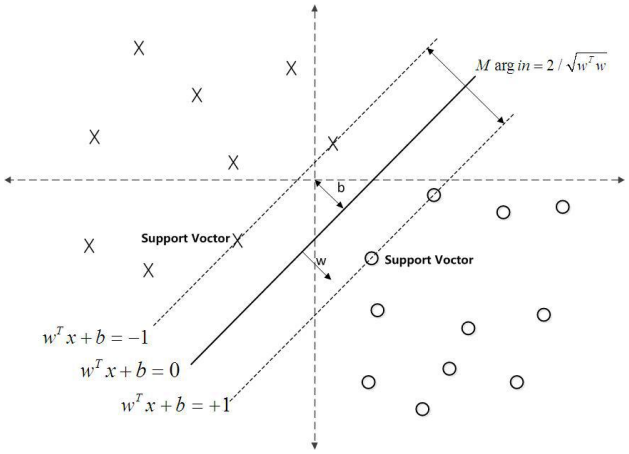
支持向量机(Support Vecor Machine,以下简称 SVM)虽然诞生只有短短的二十多年，但是自一诞生便由于它良好的分类性能席卷了机器学习领域，并牢牢压制了神经网络领域好多年。如果不考虑集成学习的算法，不考虑特定的训练数据集，在分类算法中的表现 SVM 说是排第一估计是没有什么异议的。

SVM 是一个二元分类算法，线性分类和非线性分类都支持。经过演进，现在也可以支持多元分类，同时经过扩展，也能应用于回归问题。本系列文章就对 SVM 的原理做一个总结。本篇的重点是 SVM 用于线性分类时模型和损失函数优化的一个总结。

2）支持向量

在感知机模型中，我们可以找到多个可以分类的超平面将数据分开，并且优化时希望所有的点都离超平面远。但是实际上离超平面很远的点已经被正确分类，我们让它离超平面更远并没有意义。反而我们最关心是那些离超平面很近的点，这些点很容易被误分类。如果我们可以让离超平面比较近的点尽可能的远离超平面，那么我们的分类效果会好有一些。SVM 的思想起源正起于此。

如下图所示，分离超平面为 wTx+b=0，如果所有的样本不光可以被超平面分开，还和超平面保持一定的函数距离（下图函数距离为 1），那么这样的分类超平面是比感知机的分类超平面优的。可以证明，这样的超平面只有一个。和超平面平行的保持一定的函数距离的这两个超平面对应的向量，我们定义为支持向量，如下图虚线所示。



支持向量到超平面的距离为 1/||w||2,两个支持向量之间的距离为

2/||w||2。

3)核函数

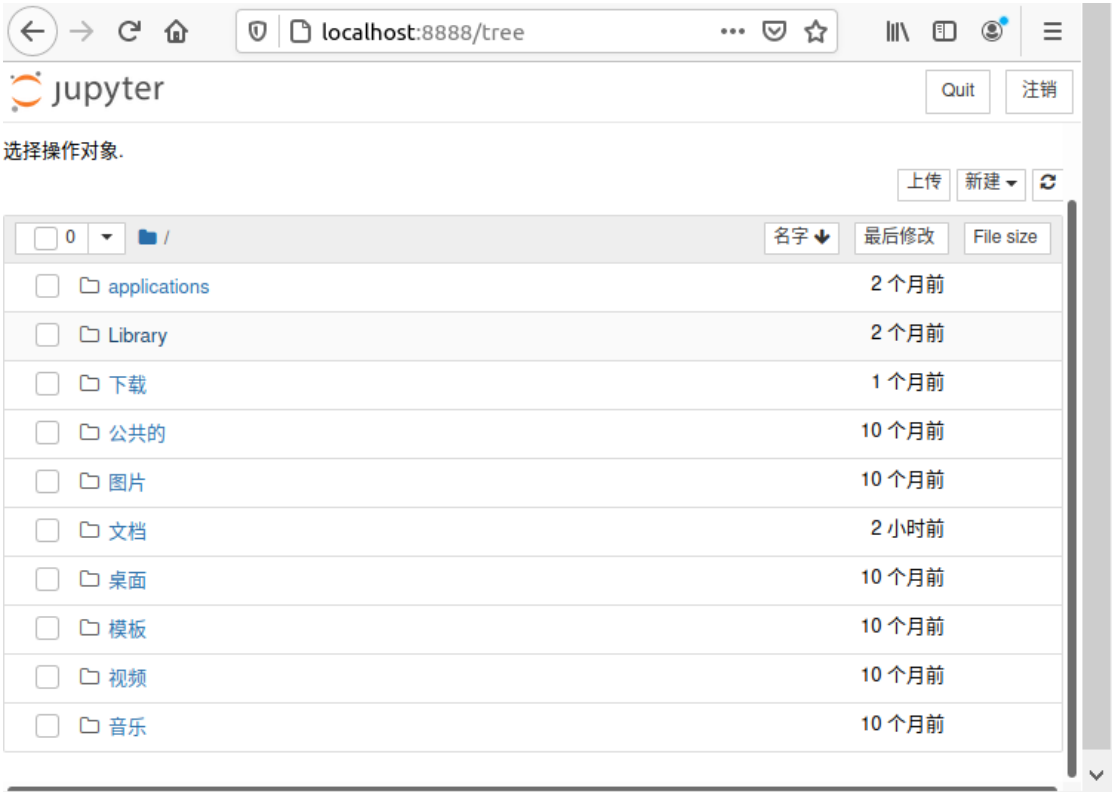
事实上，核函数的研究非常的早，要比 SVM 出现早得多，当然，将它引入 SVM 中是最近二十多年的事情。对于从低维到高维的映射，核函数不止一个。那么什么样的函数才可以当做核函数呢？这是一个有些复杂的数学问题。这里不多介绍。由于一般我们说的核函数都是正定核函数，这里我们直说明正定核函数的充分必要条件。一个函数要想成为正定核函数，必须满足他里面任何点的集合形成的 Gram 矩阵是半正定的。也就是说,对于任意的 xi∈χ,;i=1,2,3...m,K(xi,xj)对应的 Gram 矩阵 K=[K(xi,xj)]是半正定矩阵，则 K(x,z)是正定核函数。

2. 实验环境

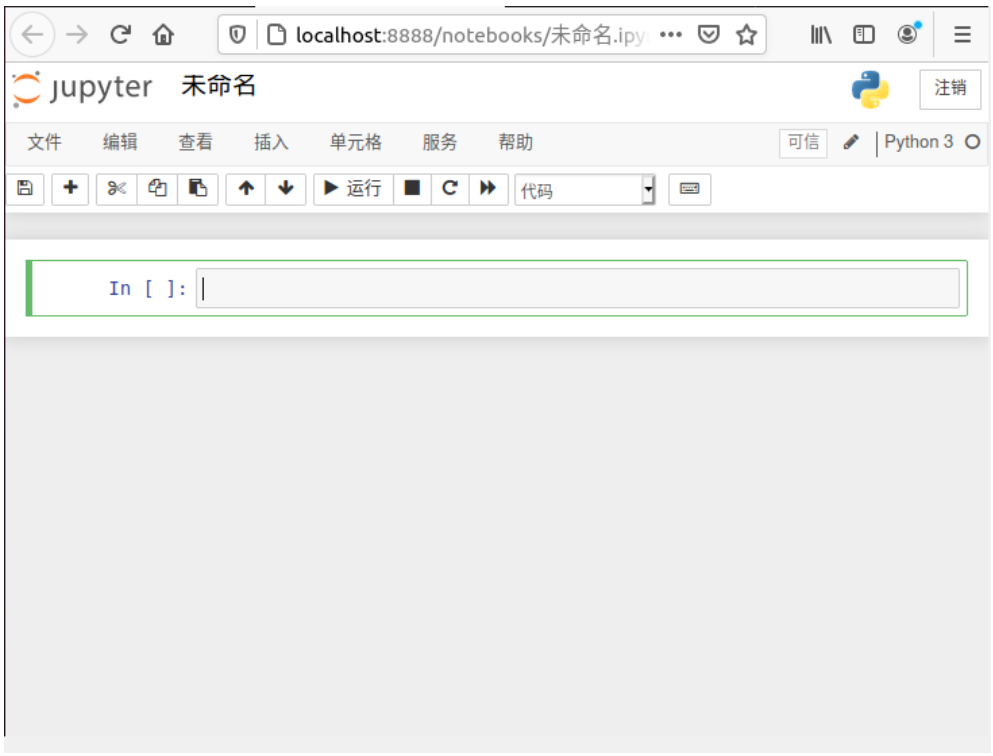
Ubuntu 20.04
Python 3.6
Jupyter notebook

3. 实验步骤

1)打开终端，然后输入 jupyter notebook，出现如下界面



2)选定特定文件夹，新建 ipynb 文件，在未命名出可重命名文件



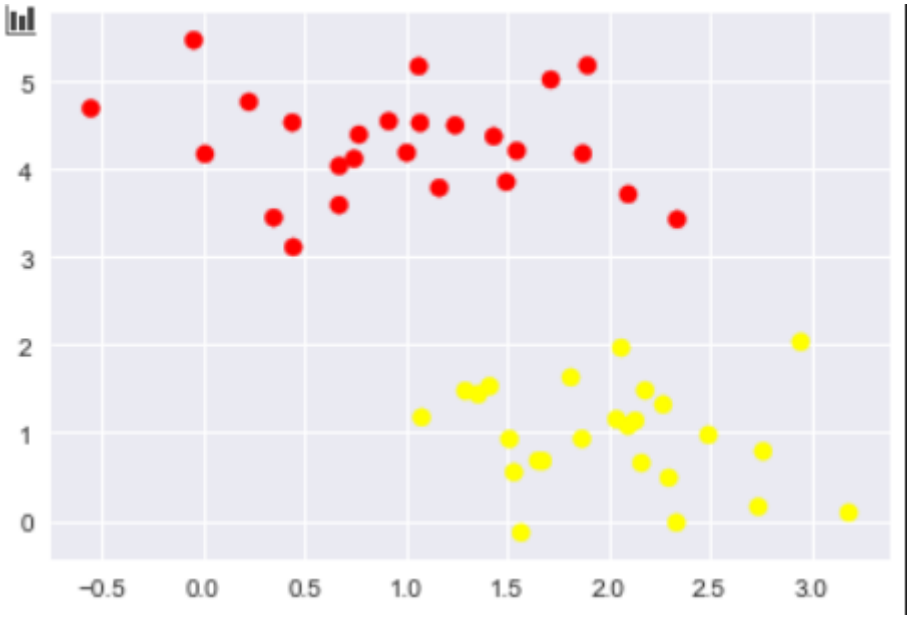
4. 实操

Step 1:导入库

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# use seaborn plotting defaults
import seaborn as sns; sns.set()
```

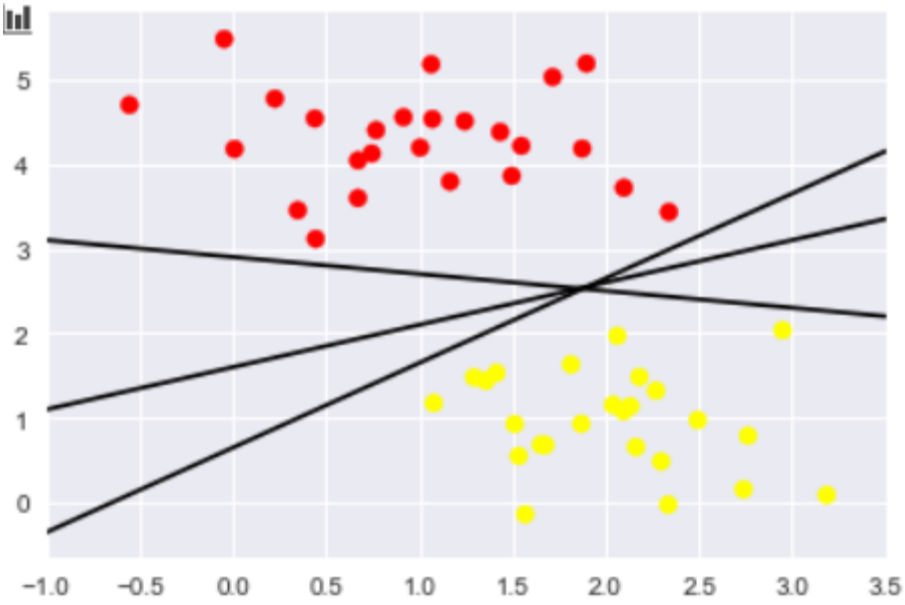
Step 2:创建模拟数据集

```
from sklearn.datasets.samples_generator import make_blobs
X, y = make_blobs(n_samples=50, centers=2,
                  random_state=0, cluster_std=0.60)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
```



Step 3:可以有多种方法分类

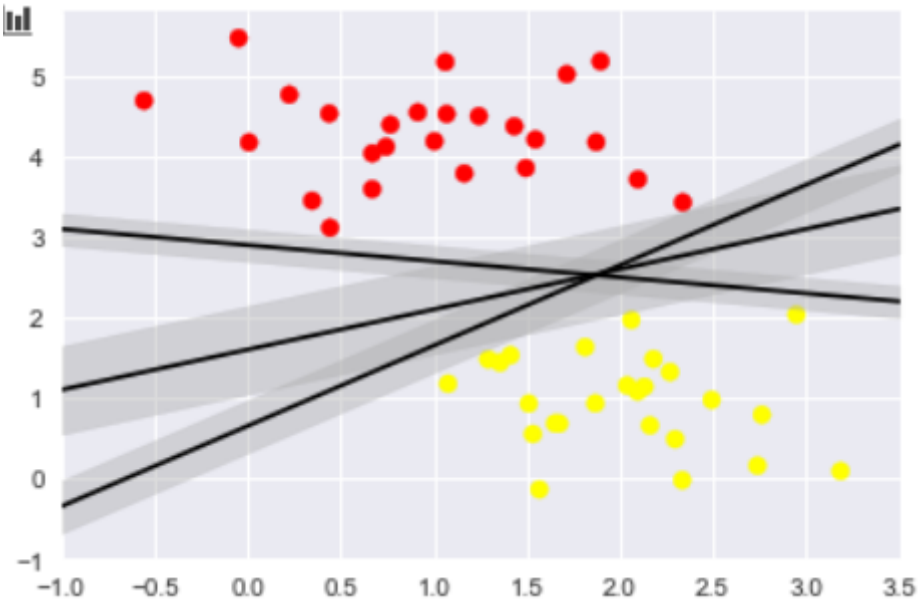
```
xfit = np.linspace(-1, 3.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')

for m, b in [(1, 0.65), (0.5, 1.6), (-0.2, 2.9)]:
    plt.plot(xfit, m * xfit + b, '-k')

plt.xlim(-1, 3.5)
```



Step 4： SVM：假想每一条分割线是有宽度的
在 SVM 的框架下，认为最宽的线为最优分割线

```
xfit = np.linspace(-1, 3.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
```

```
for m, b, d in [(1, 0.65, 0.33), (0.5, 1.6, 0.55), (-0.2, 2.9, 0.2)]:
    yfit = m * xfit + b
    plt.plot(xfit, yfit, '-k')
    plt.fill_between(xfit, yfit - d, yfit + d, edgecolor='none',
                     color='#AAAAAA', alpha=0.4)

plt.xlim(-1, 3.5)
```



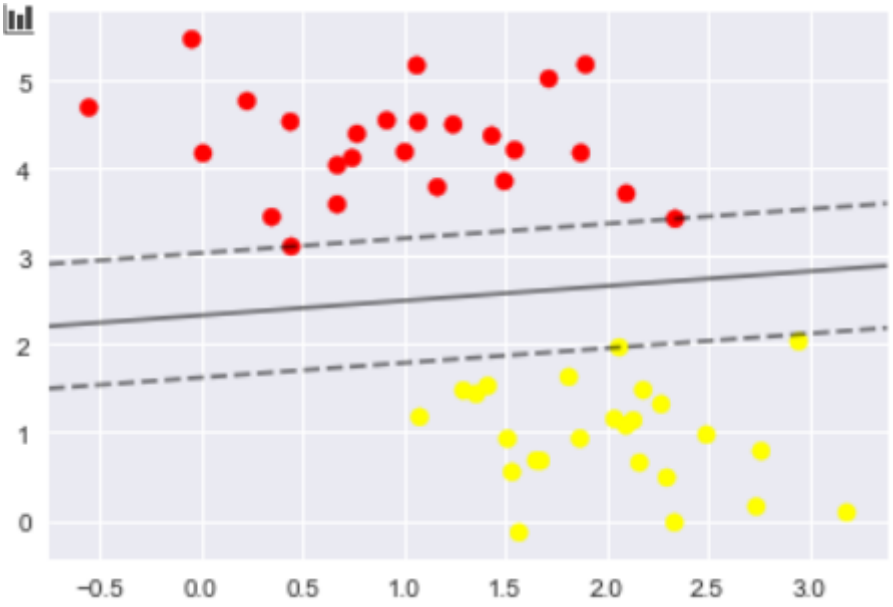
Step 5:训练 SVM

```
from sklearn.svm import SVC # "Support vector classifier"
model = SVC(kernel='linear', C=1E10)
model.fit(X, y)
```

Step 6:创建一个显示 SVM 分割线的函数

```
def plot_svc_decision_function(model, ax=None, plot_support=True):
    """Plot the decision function for a 2D SVC"""
    if ax is None:
        ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # create grid to evaluate model
    x = np.linspace(xlim[0], xlim[1], 30)
    y = np.linspace(ylim[0], ylim[1], 30)
    Y, X = np.meshgrid(y, x)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = model.decision_function(xy).reshape(X.shape)

    # plot decision boundary and margins
```

```
    ax.contour(X, Y, P, colors='k',
               levels=[-1, 0, 1], alpha=0.5,
               linestyles=['--', '-', '--'])

    # plot support vectors
    if plot_support:
        ax.scatter(model.support_vectors_[:, 0],
                   model.support_vectors_[:, 1],
                   s=300, linewidth=1, facecolors='none');
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(model)
```



Step 7:非支持向量的数据，对分割线没有影响

只有支持向量会影响分割线，如果我们添加一些非支持向量的数据，对分割线没有影响

```
def plot_svm(N=10, ax=None):
    X, y = make_blobs(n_samples=200, centers=2,
                      random_state=0, cluster_std=0.60)
    X = X[:N]
    y = y[:N]
    model = SVC(kernel='linear', C=1E10)
    model.fit(X, y)

    ax = ax or plt.gca()
    ax.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
    ax.set_xlim(-1, 4)
    ax.set_ylim(-1, 6)
```

```
    plot_svc_decision_function(model, ax)

fig, ax = plt.subplots(1, 2, figsize=(16, 6))
fig.subplots_adjust(left=0.0625, right=0.95, wspace=0.1)
for axi, N in zip(ax, [60, 120]):
    plot_svm(N, axi)
    axi.set_title('N = {0}'.format(N))
```