دانشگاه تهران

دانشکده علوم مهندسی

یادگیری ماشین

تمرین چهارم

دکتر سایه میرزایی

سپیده فاطمی خوراسگانی

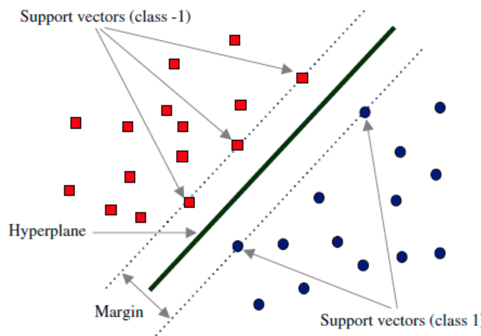
شماره دانشجویی: 810897059

بهار 00

سؤال اول)

SVM و بهترین خط متمایز کننده:

svm یک الگوریتم نظارتی می‌باشد که معمولاً برای طبقه بندی باینری (۲ کلاس) استفاده می شود.



در این روش خط جداکننده بیشترین فاصله را نسبت به داده‌ها دارد. بیشترین کاربرد آن در استفاده از کرنل ها و مرز های غیر خطی می‌باشد که در ادامه راجع به آن توضیح خواهیم داد.

معادله $w.x + b = 0$ معادله یک ابرصفحه را نشان می‌دهد که مقدار بهینه برای w همان مقدار است که تابه هزینه را min می کند. معادله ابر صفحه از رابطه زیر به دست می آید:

$$y = a*x + b$$
$$a*x + b - y = 0$$
$$X = (x,y)$$
$$W = (a, -1)$$
$$W.X + b = 0$$

درواقع برای معادله بالا اگر مقدار آن بزرگ‌تر از یک باشد در کلاس یک و اگر کوچک‌تر از یک باشد در کلاس ۱- قرار دارد. بنابراین مدل SVM را می‌توان به صورت زیر تعریف کرد:

$$svm_{model} = sign(w.x + b)$$

یا به عبارتی:

$$\mathrm{w.x}_i + b \geq 1 \quad for \quad y_i = 1$$
$$w.x_i + b \leq 1 \quad for \quad y_i = -1$$
$$combine:$$
$$y_i(w.x_i + b) - 1 \geq 0 \quad for \quad y_i = +1, -1$$

بنابر این هدف پیدا کردن بهترین ابر صفحه ای است که فاصله آن از نزدیک‌ترین داده‌های هر کلاس حداکثر باشد و از طرفی misclassification هم نداشته باشیم. یعنی برای داده جدید هم به خوبی پیش‌بینی کند.

برای به دست آوردن بهترین ابر صفحه باید پارامتر های آن یعنی w و b را به دست آوریم.

معیاانتخاب بهترین صفحه با توجه به توضیحات قبل $y_i(w \cdot x + b)$ می‌باشد که به دلیل اینکه اسکیل را رعایت کنیم آن را به صورت زیر تعریف می کنیم:

$$\gamma = y(\frac{w}{||w||} \cdot x + \frac{b}{||w||})$$

و margin یا فاصله بین نزدیک‌ترین نمونه آموزشی تا صفحه را M در نظر می گیریم:

$$M = \min_{i=1...m} y_i(\frac{w}{||w||} \cdot x + \frac{b}{||w||})$$

پس مسئله بهینه سازی زیر را باید حل کنیم:
صفحه‌ای را پیدا کنیم که فاصله Margin آن حداکثر باشد.

$$\max_{w,b} M$$
$$subject \quad to \quad \gamma_i \geq M, i = 1...m$$

و در نهایت می‌توان آن را به صورت زیر بازنویسی کرد:

$$\min_{w,b} \tfrac{1}{2}||w||^2$$
$$subject \quad to \quad y_i(w \cdot x + b) - 1 \geq 0, i = 1...m$$

در این فرمول w ماتریس ضرایب می باشد.
همان‌طور که گفته شد در فرمول بالا yi مقادیر -۱ و ۱ می‌گیرد بنابر این تابع هزینه(hinge loss function) را به صورت زیر تعریف می‌کنیم و سعی در minimize کردن آن داریم.

$$J(w) = \frac{1}{2}||w||^2 + C[\frac{1}{N}\sum_{i=1} \max(0, y_i * (w.(x_i) + b))]$$

حالا برای کمینه کردن تابع هزینه از گرادیان کاهشی استفاده می کنیم.
گرادیان وکتوری است که مشتق جزیی یا همان تغییرات را در نگه می دارد. در این روش ابتدا ضرایب را صفر قرار می‌دهیم و سپس در هر iteration آن‌ها را اپدیت می کنیم.
بنابر این برای استفاده از گرادیان کاهشی به گرادیان تابع هزینه نیاز داریم:

$$\nabla J(w) = \frac{1}{N}\sum \begin{cases} W & \text{if} \quad max(0, 1 - y_i * (w.x_i)) = 0 \\ W - Cy_ix_i & \text{otherwise} \end{cases}$$

بنابر این از تابع گرادیان بالا استفاده می‌کنیم و مقادیر W را آپدیت می کنیم.
و در نهایت معادله ابر صفحه به دست می آید. (که در دو بعد معادله خط می شود. در ادامه توضیح داده خواهد شد.)

سؤال ۲)
پارامتر c را می‌توان به عنوان عاملی دید که با استفاده از آن رفتار SVM نسبت به خطا را بررسی کرد. یعنی تعداد تجاوز ها از مارجین را کنترل می کند.
مقادیر کم c باعث مارجین بزرگـتر می‌شود که تعداد بیشتری misclassification خواهد داشت.
مقادیر زیاد c همانند hard margin عمل می کند و مارجین کوچکـتری به ما می‌دهد از طرفی به داده‌های نویزی حساس است.
پارامتر c متناسب با $\frac{1}{\lambda}$ می‌باشد یعنی عملکرد آن برای مقادیر مختل برعکسλ عمل می کند.
به طور کلی:
c بزرگ : lower bias , high variance که متناسب با λ کوچک است.
c کوچک: higher bias , low variance که متناسب با λ بزرگ است.
بهترین روش انتخاب c امتحان کردن مقادیر مختلف است می‌توان از داده‌های validation استفاده کرد و در نهایت c را انتخاب کنیم که کمترین خطا را برای داده‌های جدید دارد.

سؤال ۳)
ابتدا با استفاده از تابع generate data داده‌ها را در بازه گفته شده و به صورت رندوم ایجاد می کنیم.
داده‌های دسته اول با x1 , x2 ساخته شده‌اند و y=1 دارند
داده‌های دسته دوم با x3, x4 ساخته شده‌اند و y=2 دارند.
ابتدا داده‌ها را ترکیب می‌کنیم که یک dataframe به دست بیاوریم که بتوانیم داده‌ها را shuffle کنیم و ۸۰ درصد را برای آموزش و ۲۰ درصد را برای تست درنظر بگیریم.
این کار را با متد parse data انجام می دهیم.

```
def generate_data():
    x1 = np.random.uniform(0, 1.5, 150).reshape(150, 1)
    x2 = np.random.uniform(0, 1.5, 150).reshape(150, 1)
    x3 = np.random.uniform(2, 3.5, 150).reshape(150, 1)
    x4 = np.random.uniform(2, 3.5, 150).reshape(150, 1)

    y_1 = np.ones((150, 1))
    y_2 = np.ones((150, 1)) * -1

    x_1 = np.concatenate([x1, x2], axis=1)
    x_2 = np.concatenate([x3, x4], axis=1)

    X = np.concatenate([x_1, x_2], axis=0)
    Y = np.concatenate([y_1, y_2], axis=0)

    X = np.insert(X, 2, np.ones([300]), axis=1)
    data = np.concatenate([X, Y], axis=1)
    data = pd.DataFrame(data)
    data = data.sample(frac=1).reset_index(drop=True)
    return data
```

```python
def parse_data(data):
    train_size = int(len(data) * (1 - TEST_RATIO))

    X_train = data.loc[0:train_size-1, :FEATURE_NOM].to_numpy()
    Y_train = data.loc[0:train_size-1, FEATURE_NOM+1]

    X_test = data.loc[train_size:, :FEATURE_NOM].to_numpy()
    Y_test = data.loc[train_size:, FEATURE_NOM+1]
    return X_train, np.array(Y_train), X_test, np.array(Y_test)
```

تابع گرادیان که در سؤال قبل گفته شد را به صورت زیر پیادهسازی می کنیم:

به ازای هر داده آموزشی مقدار $1 - y_i * (w.x_i)$ را محاسبه می‌کنیم که همان distance می باشد

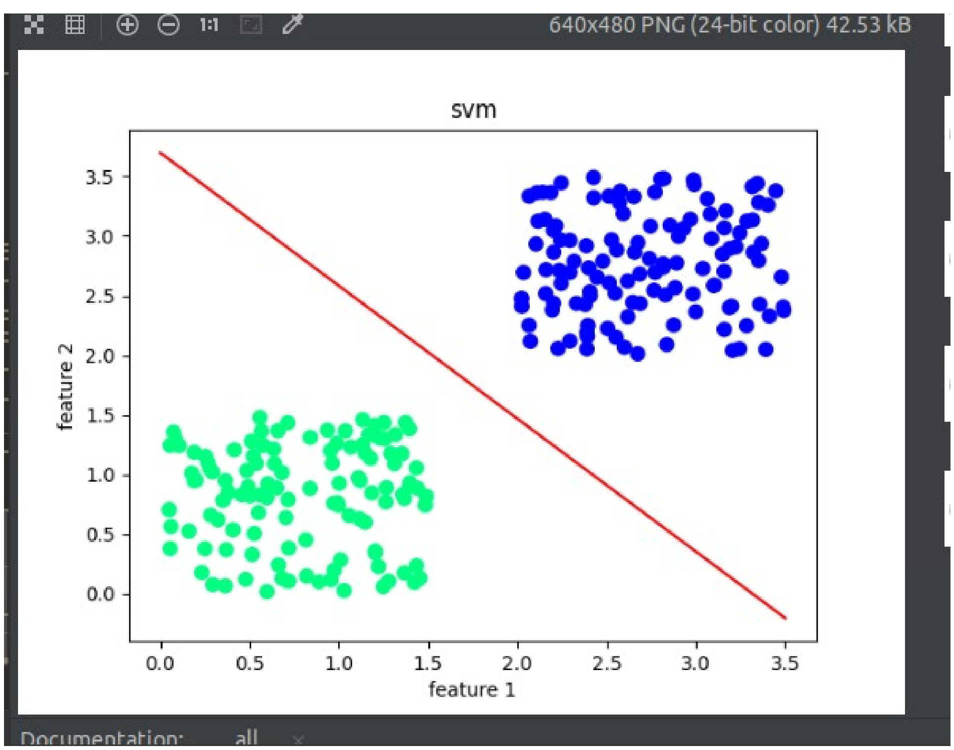
```python
def cost_gradient(W, x, y):
    sum = np.zeros(len(W))
    distance = 1 - (y * np.dot(x, W))
    if max(0, distance) == 0:
        sum += W
    else:
        sum += W - (C * y) * x
    return sum
```

با استفاده از گرادیان کاهشی وزن ها را برای هر داده آپدیت می‌کنیم و در نهایت ضریب های معادله خط را به دست می آوریم.

```python
def svm(X_train, Y_train):
    weights = np.zeros(X_train.shape[1])
    for epoch in range(1, max_epochs):
        for i in range(len(X_train)):
            temp = cost_gradient(weights, X_train[i], Y_train[i])
            weights = weights - (learning_rate * temp)
    return weights
```

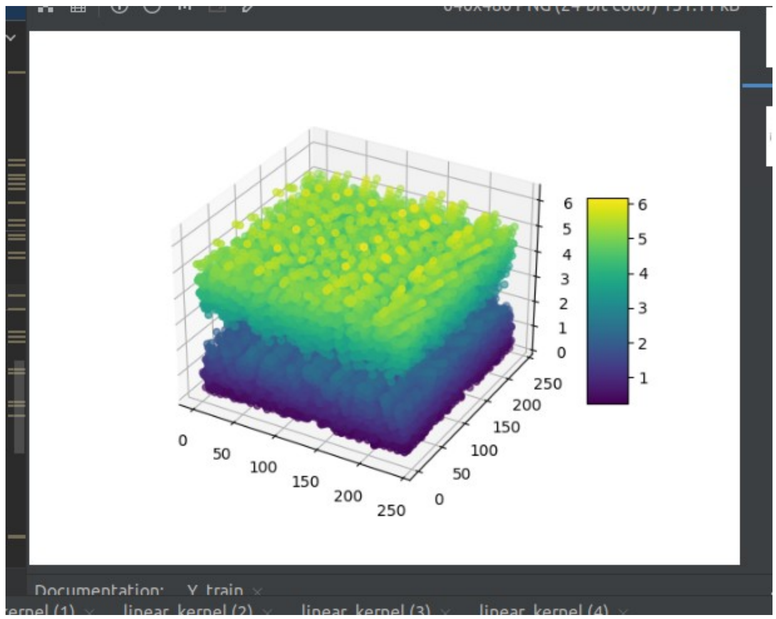
پس از رسم معادله خط به دست آمده به صورت زیر می باشد.

```python
def plot_svm(X, Y, W):
    plt.figure()
    plt.title('svm')
    plt.xlabel('feature 1')
    plt.ylabel('feature 2')
    plt.scatter(X[:, 0], X[:, 1], c=Y, s=50, cmap='winter')
    x = np.linspace(0, 3.5)
    a = -W[0] / W[1]
    b = +W[2] / W[1]
    y = a * x - b
    plt.plot(x, y, 'r')
    plt.show()
```



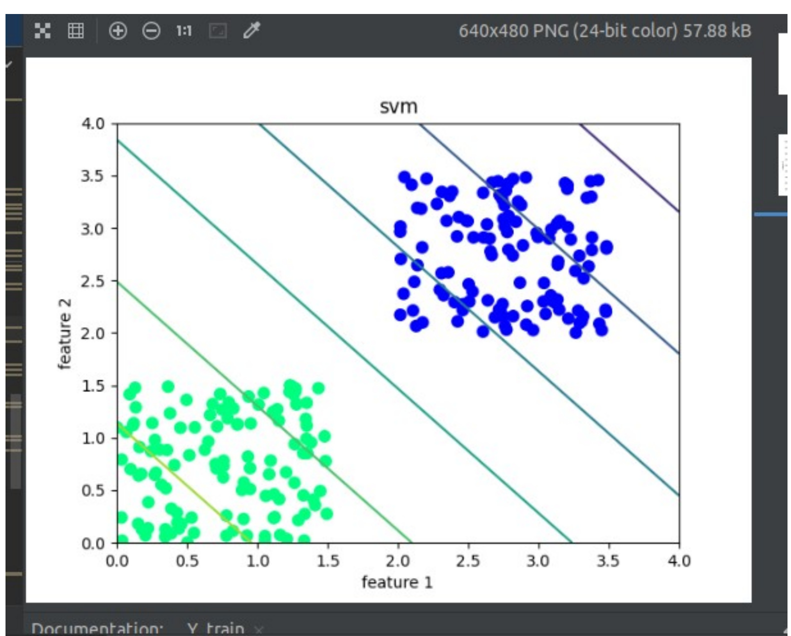
با توجه به اینکه استفاده نکردن از کرنل با استفاده از کرنل خطی تفاوتی ایجاد نمی‌کند این سؤال را با استفاده از کرنل خطی هم حل کردم که تفاوت آن در بخش‌های زیر است:
همان‌طور که مشاهده می‌شود داده‌ها را با استفاده از یک کرنل خطی به فضای سه بعدی تبدیل کردیم که صفحه‌ای می‌تواند آن‌ها را از هم جدا کند.
کرنل استفاده شده:

```
#%%
def kernel(x1, x2):
    return (np.dot(x1, x2)/4)
```



در سؤال‌های بعدی روش استفاده از کرنل توضیح داده می شود. ولی به صورت مختصر تفاوتی که در کد ایجاد می‌شود این است که این f های جدید را که با استفاده از کرنل روی داده‌های ورودی به دست آورده‌ایم به عنوان بردار ویژگی جدید در نظر می گیریم. و در نهایت یک بردار β به دست می‌آید که با ضرب در مقادیر ورودی(داده های آموزشی) ضریب معادله صفحه به دست می آید.

و در نهایت کانتور آن صفحه را روی داده‌ها رسم می کنیم:



برای داده‌های تست با توجه به $svm_{model} = sign(w.x + b)$ که در سؤال یک به دست آوردیم مقدار آن را محاسبه می‌کنیم و درواقع W.X را به دست می آوریم و اگر مثبت شد در کلاس ۱ و اگر منفی شد در کلاس ۱- قرار دارد.
سپس تعداد پیش‌بینی های درست را به دست می آوریم:

```
def accuracy(X_test, Y_test, W):
    Y_predict = []
    for i in range(X_test.shape[0]):
        Y_predict.append(np.sign(np.dot(W, X_test[i])))
    return Y_predict, np.sum(Y_test == Y_predict) / len(Y_test)
```

```
accuracy= 100.0 %

Process finished with exit code 0
```

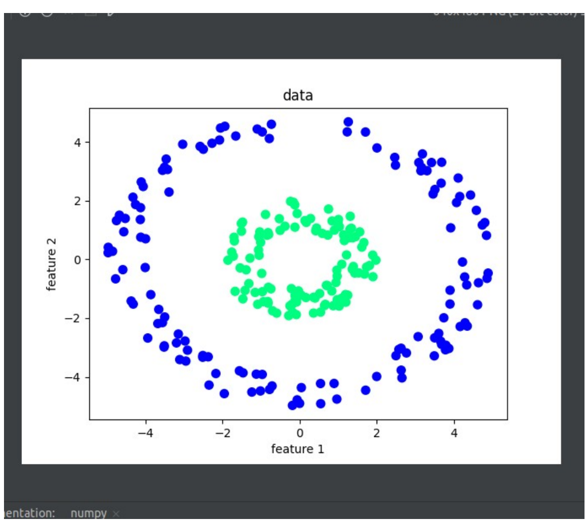
سؤال ۴)
داده‌ها را با استفاده از تابع generate data تولید می کنیم:

```
def generate_data():
    x1 = np.zeros(150).reshape(150,1)
    x2 = np.zeros(150).reshape(150,1)
    x3 = np.zeros(150).reshape(150,1)
    x4 = np.zeros(150).reshape(150,1)
    y_1 = np.ones((150, 1))
    y_2 = np.ones((150, 1)) * -1

    for i in range(150):
        x1[i], x2[i] = meteorites(1, 2)
        x3[i], x4[i] = meteorites(4, 5)

    x_1 = np.concatenate([x1, x2], axis=1)
    x_2 = np.concatenate([x3, x4], axis=1)

    X = np.concatenate([x_1, x_2], axis=0)
    Y = np.concatenate([y_1, y_2], axis=0)

    X = np.insert(X, 2, np.ones([300]), axis=1)
    data = np.concatenate([X, Y], axis=1)
    data = pd.DataFrame(data)
    data = data.sample(frac=1).reset_index(drop=True)
    return data
```

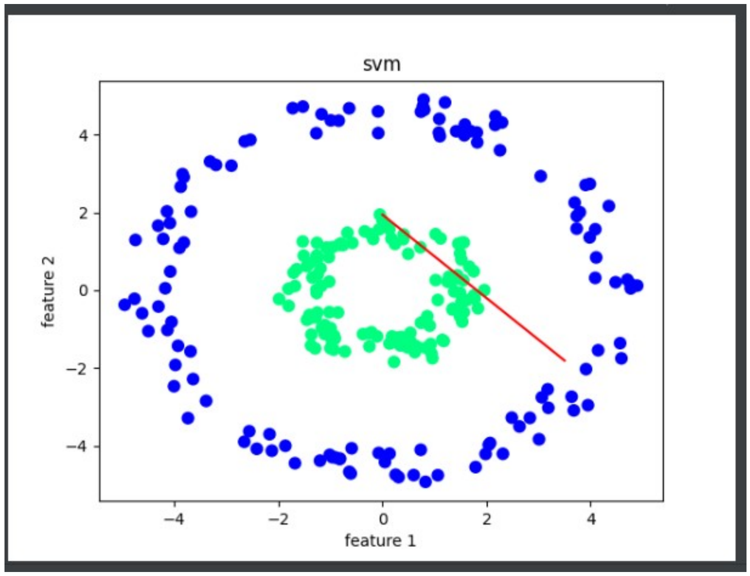
برای اینکه توزیع داده‌ها به صورت دایره‌ای باشد از تابع meteorites استفاده می کنیم:

```
def meteorites(inner, outer):
    angle = uniform(0, 2*pi)
    R = sqrt(uniform(inner * inner, outer * outer))
    x, y = R * cos(angle), R * sin(angle)
    return x, y
```

مانند سؤال قبل با استفاده از parse_data داده تست را از داده آموزشی جدا می کنیم. و داده‌های آموزشی را رسم می کنیم:

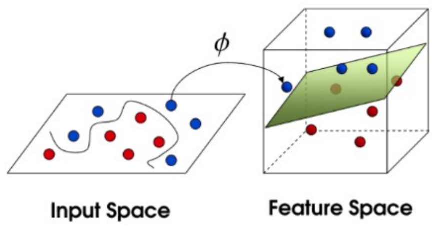


نمی‌توان مانند قسمت قبل مرز داده‌ها را به صورت خطی نشان داد.
شکل آن به صورت زیر در می آید:



سؤال ۵)
کرنل های غیر خطی:
کرنل ها را می‌توان به عنوان معیار similarity در نظر گرفت که دو ورودی می‌گیرد و خروجی آن میزان شباهت آن دو را به ما می دهد.
یکی از دلایل استفاده از کرنل این است که بدون استفاده از کرنل حجم محاسبات بسیار زیاد می‌باشد در حالی که محاسبه کرنل بسیار ساده می‌باشد و قبل از اجرای الگوریتم هم می‌توان آن را پیادهسازی کرد.
هنگامی که خود داده‌ها به صورت غیر خطی باشند یعنی مرز آن‌ها به صورت غیر خطی باشد، استفاده از کرنل های غیر خطی توصیه می شود.
به عنوان مثال برای داده‌های سؤال ۴ مرز خطی نمی‌توان برای داده‌ها تعیین کرد. ولی اگر داده‌ها را که در فضای دو بعدی قرار دارند به یک فضای بالا تری مثلاً سه بعدی نگاشت کنیم می‌توانیم ابر صفحه جدا کننده را به سادگی به دست بیاوریم.
برای مثال خطی که در ادامه سؤال ۲ حل کردم کرنل را به صورت زیر در نظر گرفتم:

$$K(x_i, x_j) = x_i \cdot x_j$$



کرنل خطی در text classification هم کاربرد دارد.
اگر ضرب داخلی xi ها را محاسبه کنیم داده‌ها را از فضای دو بعدی به فضای سه بعدی نگاشت کرده ایم.
درواقع با ساتفاده از کرنل مقادیر f را برای هر دادهx_i محاسبه کرده ایم. که شباهت داده ه را با همه داده‌های دیگر تعیین می کند.
به این صورت که:

$$f_1 = Kernel(x_i, x_1)$$
$$f_2 = Kernel(x_i, x_2)$$
.
.
.
$$f_m = Kernel(x_1, x_m)$$

درواقع برای هر داده به تعداد m (تعداد نمونه‌های آموزشی) مقدار f برای آن به دست می آوریم. و در نهایت هم چون m داده داریم، یک ماتریس $m * m$ به دست می‌آید که در کد آن را با ماتریس K نشان داده ام.
کرنل چند جمله ای:

$$K(x_i, x_j) = (x_i \cdot x_j + c)^d$$

که در فرمول بالا اگر d را یک قرار دهیم به کرنل خطی تبدیل می‌شود و مقادیر بالای d پیچیدگی مدل را زیاد می‌کند و ممکن است باعث overfit شدن مدل شود.
کرنل RBF:

$$K(x_i, x_j) = exp(-\gamma||x_i - x_j||^2)$$

Radial Basis Function که به آن کرنل گوسی هم می‌گویند برای مرز های خطی پیچیده‌تر استفاده می‌شود. پارامتر γ که تقریباً معادل با $\frac{1}{2\sigma}$ می‌باشد نشان دهنده پیچیدگی مدل می باشد. مقادیر کوچک برای γ باعث می‌شود مدل مانند کرنل خطی عمل کند و برای مقادیر بالا مدل را پیچیده می کند.

برای این سؤال چند کرنل مختلف را بررسی کردم و کرنلی که بهترین جواب را می‌داد در نظر گرفتم.

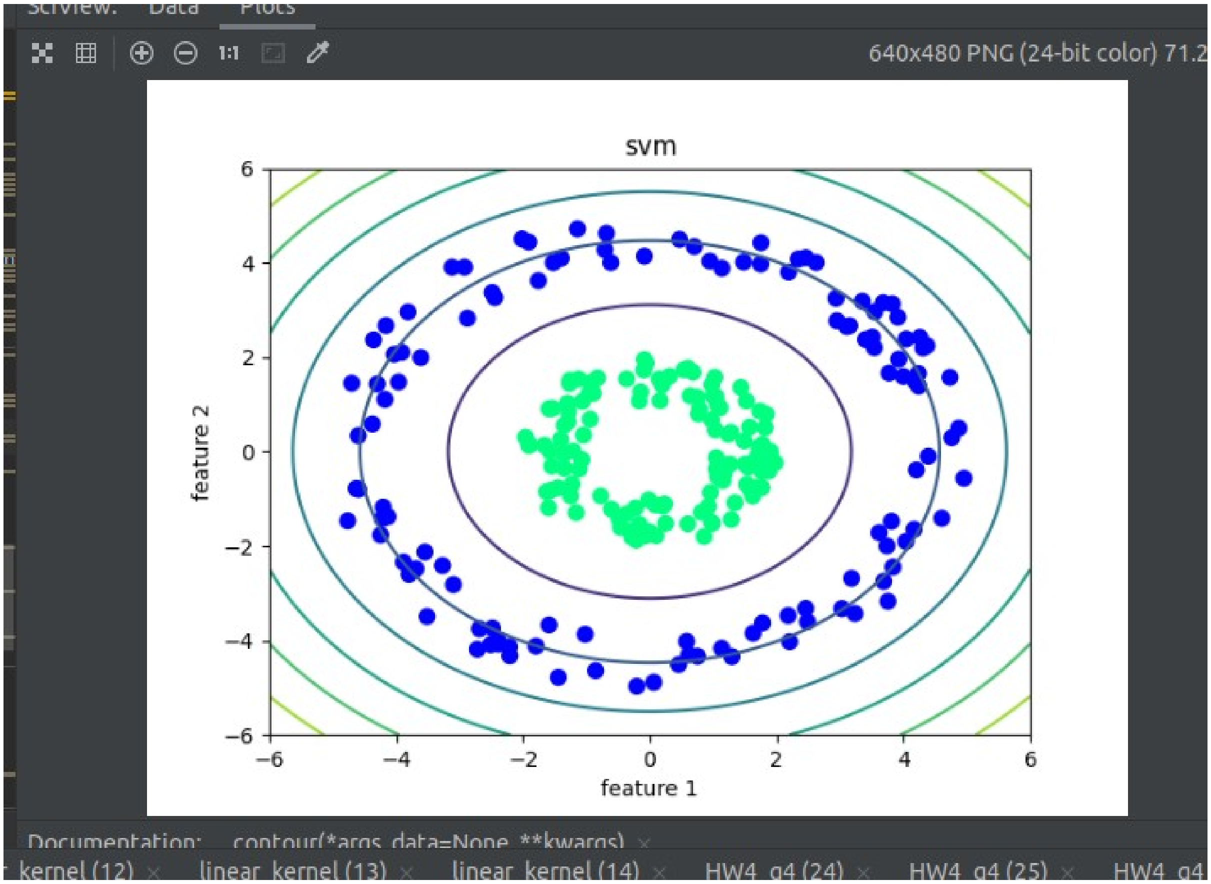
```python
def kernel(x1, x2):
    return (np.sum(x1 + x2) + 1) ** 2 #best one
    # return (np.dot(x1, x2)+0.1) ** 2 #poly
    # return np.exp(-gamma * (np.linalg.norm(x1 - x2) ** 2)) #rbf
```

بر روی همه داده‌ها for می‌زنیم و کرنل آن را با تک تک داده‌ها محاسبه می کنیم.

```python
m = len(X_train)
K = np.zeros((m, m))

for i in range(m):
    for j in range(m):
        K[i][j] = kernel(X_train[i, :], X_train.T[:, j])
```

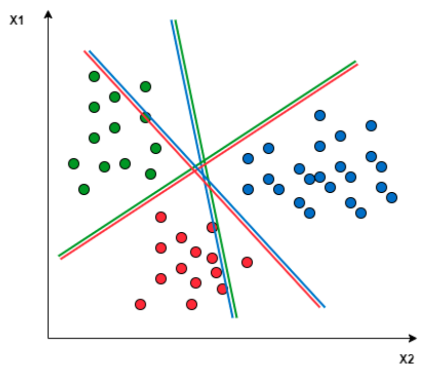
در اینجا به جای اینکه بردار X_train را به تابع svm بدهیم مقادیر W را با استفاده از این کرنل ها محاسبه می کنیم.
در نهایت مقادیر w به دست آمده ضریب های ابر صفحه می‌باشند که به دلیل اینکه به سه بعد project شده بود تصویر آن بر دو بعد یعنی contour آن را رسم می کنیم.
در این تصویر داخلی ترین دایره همان تصویر ابر صفحه بر روی ۲ بعد می باشد.

```python
def plot_svm(X, Y, W):
    W= np.array(W)
    W = np.dot(W[1:], X)
    plt.figure()
    plt.title('svm')
    plt.xlabel('feature 1')
    plt.ylabel('feature 2')
    plt.scatter(X[:, 0], X[:, 1], c=Y, s=50, cmap='winter')
    x = np.linspace(-6, 6, 100)
    y = np.linspace(-6, 6, 100)
    XX, YY = np.meshgrid(x, y)
    F = W[0]*(XX**2) + W[1]*(YY**2) - W[2]
    plt.contour(XX, YY, F)
    plt.show()
```

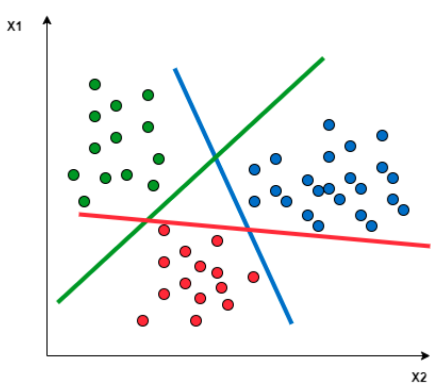

Sciview: Data Plots
640x480 PNG (24-bit color) 71.2
svm
feature 1
feature 2
Documentation: contour(*args, data=None, **kwargs)
kernel (12) linear_kernel (13) linear_kernel (14) HW4_q4 (24) HW4_q4 (25) HW4_q4

سؤال ۶) svm برای بیش از ۲ کلاس:
از svm به طور مستقیم نمی‌توان برای طبقه بندی چند کلاس استفاده کرد. ۲ روش داریم:
one-to-one approach:
ابتدا باید کلاس‌ها را به کلاس‌های ۲ تایی بشکنیم و سپس به چندین مسأله باینری تبدیل کنیم.
ایده آن به این صورت است که همانند قبل داده‌ها را به بعد بالاتری ببریم که بتوانیم آن‌ها را با ابر صفحه ها جدا کنیم. در این روش به تعداد $\frac{m(m-1)}{2}$ بار باید الگوریتم svm را اجرا کنیم. (m تعداد کلاس‌ها می باشد)

در شکل رو به رو برای هر کلاس به ازای تمام کلاس‌ها مرز خطی داریم.



One-to-rest:
در این روش هر کلاس را از مجموع بقیه کلاس‌ها جدا می کنیم. در این روش به تعداد کلاس‌ها باید svm بزنیم.



اگر ویژگی‌ها بیشتر از ۲ تا باشند مانند قبل از کرنل های غیر خطی استفاده می‌کنیم و مرز های غیر خطی را از یکی از روش‌های بالا به دست می آوریم.

سؤال ۸)

```python
def load_data(file):
    data = pd.read_csv(file)
    cleanup = {"class": {"Iris-virginica":0, "Iris-versicolor": 1, "Iris-setosa":2}}
    data= data.replace(cleanup)
    # print(data['class'].value_counts())
    x1 = data['sepallength'].to_numpy().reshape(150, 1)
    x2 = data['sepalwidth'].to_numpy().reshape(150, 1)
    y = data['class'].to_numpy().reshape(150, 1)
    x = np.concatenate([x1, x2], axis=1)
    x = np.insert(x, 2, np.ones([150]), axis=1)
    new_data = np.concatenate([x, y], axis=1)
    new_data = pd.DataFrame(new_data)
    new_data = new_data.sample(frac=1).reset_index(drop=True)
    return new_data
```