Worksheet 16

Name: Jeong Yong Yang, Junho Son UID: U95912941, U64222022

Topics

- Support Vector Machines (Non-linear case)

Support Vector Machines

Follow along in class to implement the perceptron algorithm and create an animation of the algorithm.

a) As we saw in class, the form

$$w^T x + b = 0$$

while simple, does not expose the inner product `<x_i, x_j>` which we know `w` depends on, having done the math. This is critical to applying the "kernel trick" which allows for learning non-linear decision boundaries. Let's modify the above algorithm to use the form

$$\sum_i \alpha_i < x_i, x > + b = 0$$

```python
import numpy as np
from PIL import Image as im
import matplotlib.pyplot as plt
import sklearn.datasets as datasets

TEMPFILE = "temp.png"
CENTERS = [[0, 1], [1, 0]]

epochs = 100
learning_rate = .05
expanding_rate = .99
retracting_rate = 1.1

X, labels = datasets.make_blobs(n_samples=10, centers=CENTERS, cluster_std=0.2, random_state=0
Y = np.array(list(map(lambda x : -1 if x == 0 else 1, labels.tolist())))

alpha_i = np.zeros((len(X),))
b = 0

def snap(x, alpha_i, b, error):
    # create a mesh to plot in
    h = .01  # step size in the mesh
    x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
    y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    meshData = np.c_[xx.ravel(), yy.ravel()]
    cs = np.array([x for x in 'gb'])
    fig, ax = plt.subplots()
    ax.scatter(X[:,0],X[:,1],color=cs[labels].tolist(), s=50, alpha=0.8)

    if error:
        ax.add_patch(plt.Circle((x[0], x[1]), .12, color='r',fill=False))
    else:
        ax.add_patch(plt.Circle((x[0], x[1]), .12, color='y',fill=False))

    Z = predict_many(alpha_i, b, meshData)
    Z = np.array([0 if z <=0 else 1 for z in Z]).reshape(xx.shape)
    ax.contourf(xx, yy, Z, alpha=.5, cmap=plt.cm.Paired)
    fig.savefig(TEMPFILE)
    plt.close()
    return im.fromarray(np.asarray(im.open(TEMPFILE)))

def predict_many(alpha_i, b, Z):
    res = []
    for i in range(len(Z)):
        res.append(predict(alpha_i, b, Z[i]))
    return np.array(res)

def predict(alpha_i, b, x):
    wx = 0
    for a in range(len(x)):
      wx += alpha_i[a] * np.dot(x[a], x[0])
      break
    return wx + b

images = []
for _ in range(epochs):
    # pick a point from X at random
    i = np.random.randint(0, len(X))
    error = False
    x, y = X[i], Y[i]
    ypred = predict(alpha_i, b, x)

    if (ypred.all() > 0 and y > 0) or (ypred.all() < 0 and y < 0):
      error = True
      print(True)
      if ypred.all() < 1 and ypred.all() > -1:
        alpha_i[i] += y * learning_rate
        alpha_i = alpha_i * retracting_rate
        b += y * learning_rate * retracting_rate
```

```
        else:
          alpha_i = alpha_i * expanding_rate
          b += expanding_rate


    images.append(snap(x, alpha_i, b, error))




images[0].save(
    'svm_dual.gif',
    optimize=False,
    save_all=True,
    append_images=images[1:],
    loop=0,
    duration=100
```

```
import numpy as np
from PIL import Image as im
import matplotlib.pyplot as plt
import sklearn.datasets as datasets


CENTERS = [(1, 1), (-1, -1), (1, -1), (-1, 1)]
X, _ = datasets.make_blobs(n_samples=10, centers=CENTERS, cluster_std=0.6, random_state=0)
labels = [0, 1, 1, 0, 1, 1, 1, 0, 1, 0]
Y = np.array([-1 if x == 0 else 1 for x in labels])

epochs = 100
learning_rate = .05
expanding_rate = .99
retracting_rate = 1.1

TEMPFILE = "temp2.png"
alpha_i = np.zeros((len(X),))
b = 0

def snap(x, alpha_i, b, error):
    # create a mesh to plot in
    h = .01  # step size in the mesh
    x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
    y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    meshData = np.c_[xx.ravel(), yy.ravel()]
    cs = np.array([x for x in 'gb'])
    fig, ax = plt.subplots()
    ax.scatter(X[:,0],X[:,1],color=cs[labels].tolist(), s=50, alpha=0.8)

    if error:
        ax.add_patch(plt.Circle((x[0], x[1]), .12, color='r',fill=False))
    else:
        ax.add_patch(plt.Circle((x[0], x[1]), .12, color='y',fill=False))

    Z = predict_many(alpha_i, b, meshData)
    Z = np.array([0 if z <=0 else 1 for z in Z]).reshape(xx.shape)
    ax.contourf(xx, yy, Z, alpha=.5, cmap=plt.cm.Paired)
    fig.savefig(TEMPFILE)
    plt.close()
    return im.fromarray(np.asarray(im.open(TEMPFILE)))

def predict_many(alpha_i, b, Z):
    res = []
    for i in range(len(Z)):
        res.append(predict(alpha_i, b, Z[i]))
    return np.array(res)

def predict(alpha_i, b, x):
    wx = 0
    for a in range(len(x)):
```

```
        wx += alpha_i[a] * np.dot(x[a], x[0])
        break
    return wx + b


images = []
for _ in range(epochs):
    # pick a point from X at random
    i = np.random.randint(0, len(X))
    error = False
    x, y = X[i], Y[i]
    ypred = predict(alpha_i, b, x)

    if (ypred.all() > 0 and y > 0) or (ypred.all() < 0 and y < 0):
      error = True
      print(True)
      if ypred.all() < 1 and ypred.all() > -1:
        alpha_i[i] += y * learning_rate
        alpha_i = alpha_i * retracting_rate
        b += y * learning_rate * retracting_rate
      else:
        alpha_i = alpha_i * expanding_rate
        b += expanding_rate


    images.append(snap(x, alpha_i, b, error))




images[0].save(
    'svm_dual2.gif',
    optimize=False,
    save_all=True,
    append_images=images[1:],
    loop=0,
    duration=100
)
```

Write a configurable kernel function to apply in lieu of the dot product. Try it out on a dataset that is not linearly separable.

```
def polynomial(x_i, x_j, c, n):
    return (np.dot(x_i, x_j) + c) ** n

def predict(alpha_i, b, x):
    wx = 0
    for a in range(len(x)):
      wx += alpha_i[a] * polynomial(X[a], x, C, N)
      break
    return wx + b
```

b) Assume we fit an SVM using a polynomial Kernel function and it seems to overfit the data. How would you adjust the tuning parameter `n` of the kernel function?

We can reduce the degree, the value of n. As the value of n increases, the decision boundaries become more complexed and therefore, the model can overfit the dataset. If we decrease the value of n, it might not cause overfitting of the dataset.

c) Assume we fit an SVM using a RBF Kernel function and it seems to underfit the data. How would you adjust the tuning parameter `sigma` of the kernel function?

We can decrease the value of sigma since it can allow the model to become more complex and therefore, the model would be able to avoid underfitting.

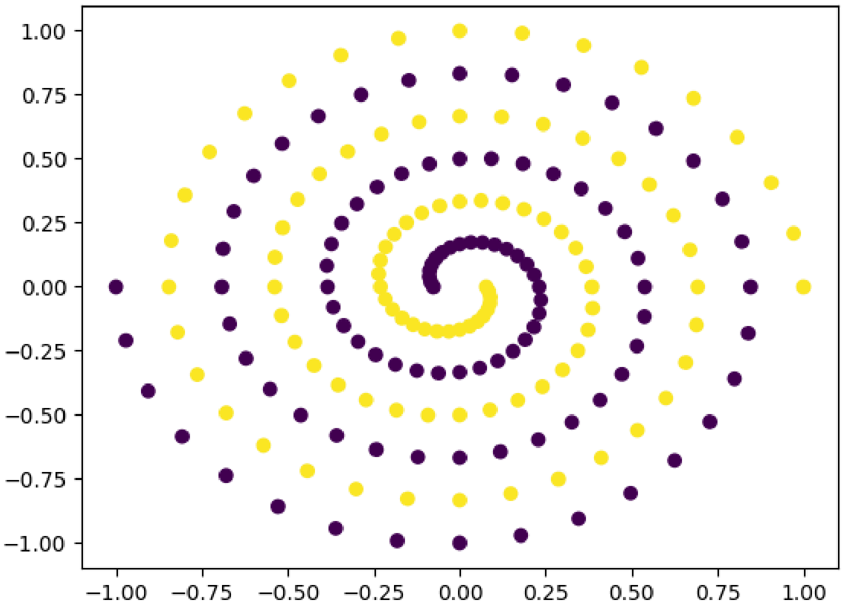
d) Tune the parameter of a specific Kernel function, to fit an SVM (using your code above) to the following dataset:

```python
import numpy as np
import matplotlib.pyplot as plt

data = np.loadtxt("spiral.data")
X, Y = data[:, :2], data[:, 2]

plt.scatter(X[:,0], X[:,1], c=Y)
```

```
<matplotlib.collections.PathCollection at 0x7d088472d420>
```




```python
C = 1
N = 3

epochs = 100
learning_rate = .05
expanding_rate = .99
retracting_rate = 1.1

TEMPFILE = "temp3.png"
alpha_i = np.zeros((len(X),))
b = 0

def snap(x, alpha_i, b, error):
    # create a mesh to plot in
    h = .01  # step size in the mesh
    x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
    y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    meshData = np.c_[xx.ravel(), yy.ravel()]
    cs = np.array([x for x in 'gb'])
    fig, ax = plt.subplots()
    ax.scatter(X[:,0],X[:,1],color=cs[labels].tolist(), s=50, alpha=0.8)

    if error:
        ax.add_patch(plt.Circle((x[0], x[1]), .12, color='r',fill=False))
    else:
        ax.add_patch(plt.Circle((x[0], x[1]), .12, color='y',fill=False))

    Z = predict_many(alpha_i, b, meshData)
    Z = np.array([0 if z <=0 else 1 for z in Z]).reshape(xx.shape)
    ax.contourf(xx, yy, Z, alpha=.5, cmap=plt.cm.Paired)
    fig.savefig(TEMPFILE)
    plt.close()
    return im.fromarray(np.asarray(im.open(TEMPFILE)))

def predict_many(alpha_i, b, Z):
    res = []
    for i in range(len(Z)):
```

```
    for i in range(len(Z)):
        res.append(predict(alpha_i, b, Z[i]))
    return np.array(res)

def polynomial(x_i, x_j, c, n):
    return (np.dot(x_i, x_j) + c) ** n

def predict(alpha_i, b, x):
    wx = 0
    for a in range(len(x)):
      wx += alpha_i[a] * polynomial(X[a], x, C, N)
      break
    return wx + b
```

```

images = []
for _ in range(epochs):
    # pick a point from X at random
    i = np.random.randint(0, len(X))
    error = False
    x, y = X[i], Y[i]
    ypred = predict(alpha_i, b, x)
```