**Module 5 Lab. Tutorial 1**

**[1]** print(\_\_doc\_\_)

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.colors import Normalize

from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler

from sklearn.datasets import load\_iris

from sklearn.model\_selection import StratifiedShuffleSplit

from sklearn.model\_selection import GridSearchCV

# Utility function to move the midpoint of a colormap to be around

# the values of interest.

class MidpointNormalize(Normalize):

def \_\_init\_\_(self, vmin=None, vmax=None, midpoint=None, clip=False):

self.midpoint = midpoint

Normalize.\_\_init\_\_(self, vmin, vmax, clip)

def \_\_call\_\_(self, value, clip=None):

x, y = [self.vmin, self.midpoint, self.vmax], [0, 0.5, 1]

return np.ma.masked\_array(np.interp(value, x, y))

# #############################################################################

# Load and prepare data set

#

# dataset for grid search

iris = load\_iris()

X = iris.data

y = iris.target

# Dataset for decision function visualization: we only keep the first two

# features in X and sub-sample the dataset to keep only 2 classes and

# make it a binary classification problem.

X\_2d = X[:, :2]

X\_2d = X\_2d[y > 0]

y\_2d = y[y > 0]

y\_2d -= 1

# It is usually a good idea to scale the data for SVM training.

# We are cheating a bit in this example in scaling all of the data,

# instead of fitting the transformation on the training set and

# just applying it on the test set.

scaler = StandardScaler()

X = scaler.fit\_transform(X)

X\_2d = scaler.fit\_transform(X\_2d)

# #############################################################################

# Train classifiers

#

# For an initial search, a logarithmic grid with basis

# 10 is often helpful. Using a basis of 2, a finer

# tuning can be achieved but at a much higher cost.

C\_range = np.logspace(-2, 10, 13)

gamma\_range = np.logspace(-9, 3, 13)

param\_grid = dict(gamma=gamma\_range, C=C\_range)

cv = StratifiedShuffleSplit(n\_splits=5, test\_size=0.2, random\_state=42)

grid = GridSearchCV(SVC(), param\_grid=param\_grid, cv=cv)

grid.fit(X, y)

print("The best parameters are %s with a score of %0.2f"

% (grid.best\_params\_, grid.best\_score\_))

# Now we need to fit a classifier for all parameters in the 2d version

# (we use a smaller set of parameters here because it takes a while to train)

C\_2d\_range = [1e-2, 1, 1e2]

gamma\_2d\_range = [1e-1, 1, 1e1]

classifiers = []

for C in C\_2d\_range:

for gamma in gamma\_2d\_range:

clf = SVC(C=C, gamma=gamma)

clf.fit(X\_2d, y\_2d)

classifiers.append((C, gamma, clf))

# #############################################################################

# Visualization

#

# draw visualization of parameter effects

plt.figure(figsize=(8, 6))

xx, yy = np.meshgrid(np.linspace(-3, 3, 200), np.linspace(-3, 3, 200))

for (k, (C, gamma, clf)) in enumerate(classifiers):

# evaluate decision function in a grid

Z = clf.decision\_function(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

# visualize decision function for these parameters

plt.subplot(len(C\_2d\_range), len(gamma\_2d\_range), k + 1)

plt.title("gamma=10^%d, C=10^%d" % (np.log10(gamma), np.log10(C)),

size='medium')

# visualize parameter's effect on decision function

plt.pcolormesh(xx, yy, -Z, cmap=plt.cm.RdBu)

plt.scatter(X\_2d[:, 0], X\_2d[:, 1], c=y\_2d, cmap=plt.cm.RdBu\_r,

edgecolors='k')

plt.xticks(())

plt.yticks(())

plt.axis('tight')

scores = grid.cv\_results\_['mean\_test\_score'].reshape(len(C\_range),

len(gamma\_range))

# Draw heatmap of the validation accuracy as a function of gamma and C

#

# The score are encoded as colors with the hot colormap which varies from dark

# red to bright yellow. As the most interesting scores are all located in the

# 0.92 to 0.97 range we use a custom normalizer to set the mid-point to 0.92 so

# as to make it easier to visualize the small variations of score values in the

# interesting range while not brutally collapsing all the low score values to

# the same color.

plt.figure(figsize=(8, 6))

plt.subplots\_adjust(left=.2, right=0.95, bottom=0.15, top=0.95)

plt.imshow(scores, interpolation='nearest', cmap=plt.cm.hot,

norm=MidpointNormalize(vmin=0.2, midpoint=0.92))

plt.xlabel('gamma')

plt.ylabel('C')

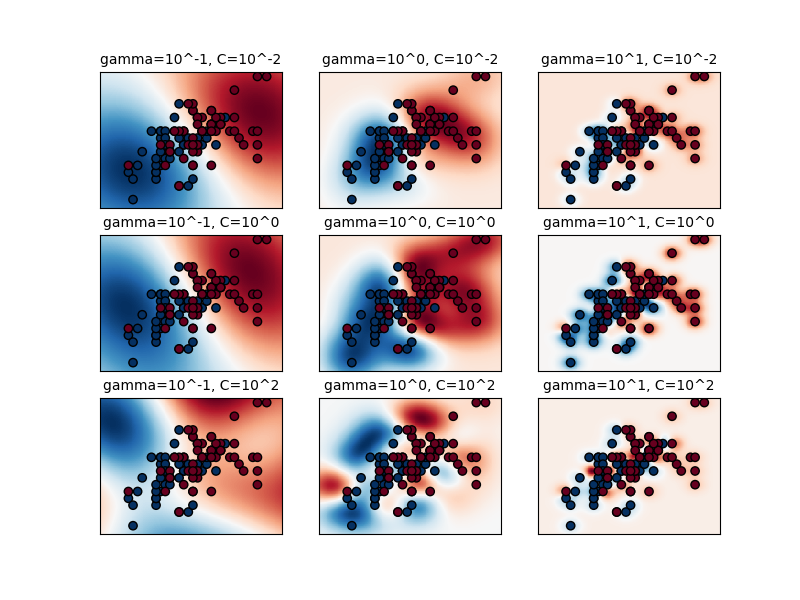
plt.colorbar()

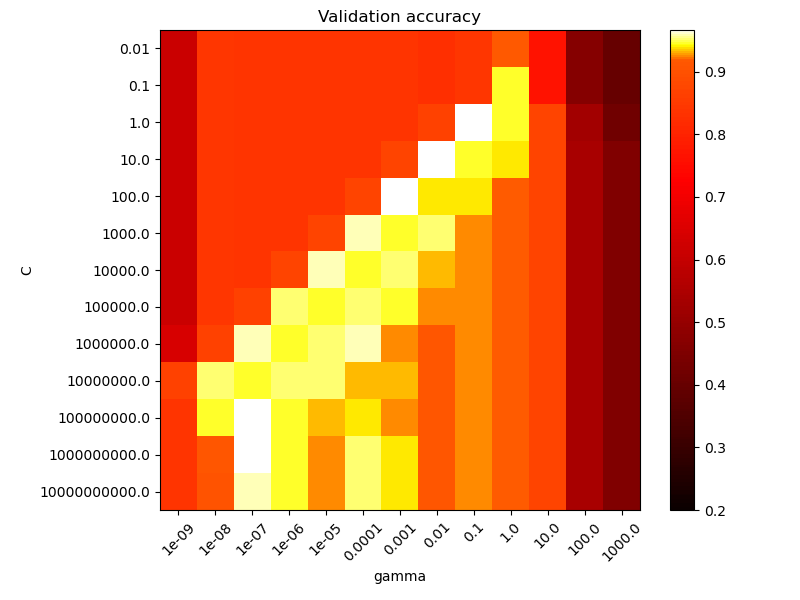
plt.xticks(np.arange(len(gamma\_range)), gamma\_range, rotation=45)

plt.yticks(np.arange(len(C\_range)), C\_range)

plt.title('Validation accuracy')

plt.show()





**Module 5 Lab. Tutorial 2**

**[2]** def gaussian\_rbf(x, landmark, gamma):

return np.exp(-gamma \* np.linalg.norm(x - landmark, axis=1)\*\*2)

gamma = 0.3

x1s = np.linspace(-4.5, 4.5, 200).reshape(-1, 1)

x2s = gaussian\_rbf(x1s, -2, gamma)

x3s = gaussian\_rbf(x1s, 1, gamma)

XK = np.c\_[gaussian\_rbf(X1D, -2, gamma), gaussian\_rbf(X1D, 1, gamma)]

yk = np.array([0, 0, 1, 1, 1, 1, 1, 0, 0])

plt.figure(figsize=(10.5, 4))

plt.subplot(121)

plt.grid(True, which='both')

plt.axhline(y=0, color='k')

plt.scatter(x=[-2, 1], y=[0, 0], s=150, alpha=0.5, c="red")

plt.plot(X1D[:, 0][yk==0], np.zeros(4), "bs")

plt.plot(X1D[:, 0][yk==1], np.zeros(5), "g^")

plt.plot(x1s, x2s, "g--")

plt.plot(x1s, x3s, "b:")

plt.gca().get\_yaxis().set\_ticks([0, 0.25, 0.5, 0.75, 1])

plt.xlabel(r"$x\_1$", fontsize=20)

plt.ylabel(r"Similarity", fontsize=14)

plt.annotate(r'$\mathbf{x}$',

xy=(X1D[3, 0], 0),

xytext=(-0.5, 0.20),

ha="center",

arrowprops=dict(facecolor='black', shrink=0.1),

fontsize=18,

)

plt.text(-2, 0.9, "$x\_2$", ha="center", fontsize=20)

plt.text(1, 0.9, "$x\_3$", ha="center", fontsize=20)

plt.axis([-4.5, 4.5, -0.1, 1.1])

plt.subplot(122)

plt.grid(True, which='both')

plt.axhline(y=0, color='k')

plt.axvline(x=0, color='k')

plt.plot(XK[:, 0][yk==0], XK[:, 1][yk==0], "bs")

plt.plot(XK[:, 0][yk==1], XK[:, 1][yk==1], "g^")

plt.xlabel(r"$x\_2$", fontsize=20)

plt.ylabel(r"$x\_3$  ", fontsize=20, rotation=0)

plt.annotate(r'$\phi\left(\mathbf{x}\right)$',

xy=(XK[3, 0], XK[3, 1]),

xytext=(0.65, 0.50),

ha="center",

arrowprops=dict(facecolor='black', shrink=0.1),

fontsize=18,

)

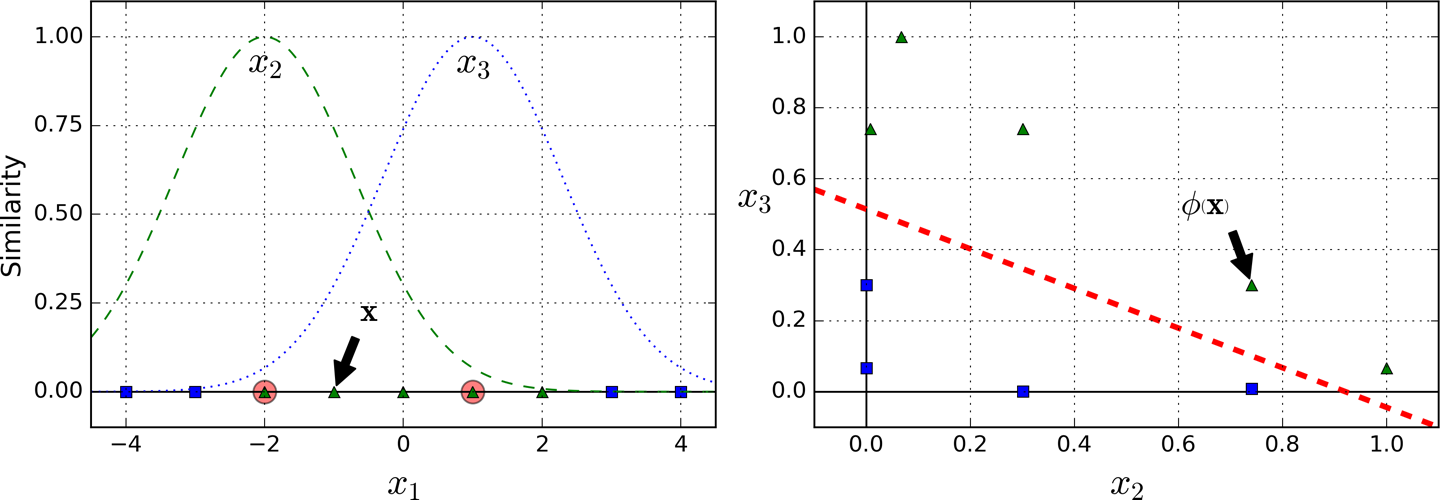
plt.plot([-0.1, 1.1], [0.57, -0.1], "r--", linewidth=3)

plt.axis([-0.1, 1.1, -0.1, 1.1])

plt.subplots\_adjust(right=1)

save\_fig("kernel\_method\_plot")

plt.show()



**[3]** x1\_example = X1D[3, 0]

for landmark in (-2, 1):

k = gaussian\_rbf(np.array([[x1\_example]]), np.array([[landmark]]), gamma)

print("Phi({}, {}) = {}".format(x1\_example, landmark, k))

**[4]** rbf\_kernel\_svm\_clf = Pipeline([

("scaler", StandardScaler()),

("svm\_clf", SVC(kernel="rbf", gamma=5, C=0.001))

])

rbf\_kernel\_svm\_clf.fit(X, y)

**[5]** from sklearn.svm import SVC

gamma1, gamma2 = 0.1, 5

C1, C2 = 0.001, 1000

hyperparams = (gamma1, C1), (gamma1, C2), (gamma2, C1), (gamma2, C2)

svm\_clfs = []

for gamma, C in hyperparams:

rbf\_kernel\_svm\_clf = Pipeline([

("scaler", StandardScaler()),

("svm\_clf", SVC(kernel="rbf", gamma=gamma, C=C))

])

rbf\_kernel\_svm\_clf.fit(X, y)

svm\_clfs.append(rbf\_kernel\_svm\_clf)

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10.5, 7), sharex=True, sharey=True)

for i, svm\_clf in enumerate(svm\_clfs):

plt.sca(axes[i // 2, i % 2])

plot\_predictions(svm\_clf, [-1.5, 2.45, -1, 1.5])

plot\_dataset(X, y, [-1.5, 2.45, -1, 1.5])

gamma, C = hyperparams[i]

plt.title(r"$\gamma = {}, C = {}$".format(gamma, C), fontsize=16)

if i in (0, 1):

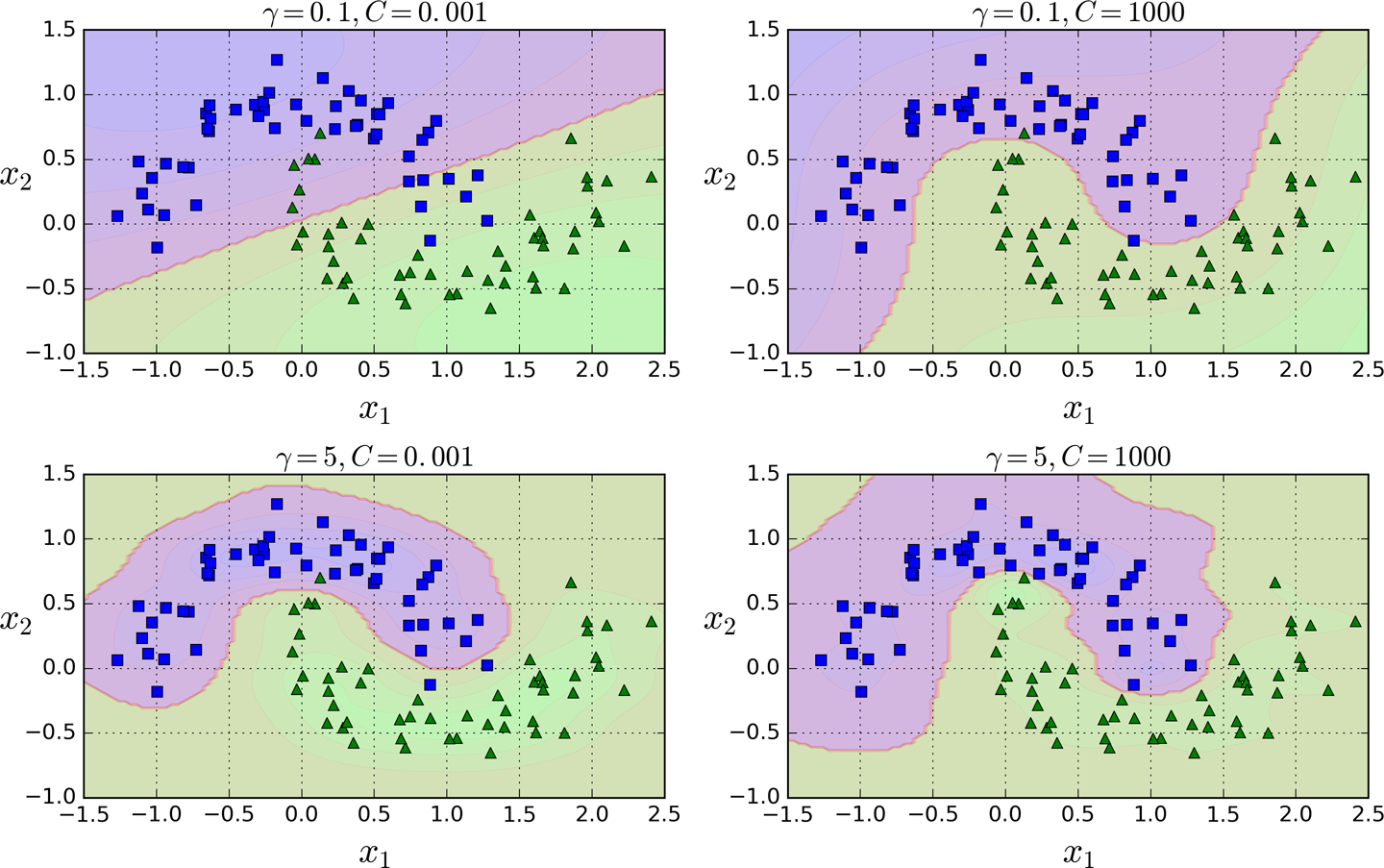
plt.xlabel("")

if i in (1, 3):

plt.ylabel("")

save\_fig("moons\_rbf\_svc\_plot")

plt.show()



**[6]** np.random.seed(42)

m = 50

X = 2 \* np.random.rand(m, 1)

y = (4 + 3 \* X + np.random.randn(m, 1)).ravel()

**[7]** from sklearn.svm import LinearSVR

svm\_reg = LinearSVR(epsilon=1.5, random\_state=42)

svm\_reg.fit(X, y)

**[8]** svm\_reg1 = LinearSVR(epsilon=1.5, random\_state=42)

svm\_reg2 = LinearSVR(epsilon=0.5, random\_state=42)

svm\_reg1.fit(X, y)

svm\_reg2.fit(X, y)

def find\_support\_vectors(svm\_reg, X, y):

y\_pred = svm\_reg.predict(X)

off\_margin = (np.abs(y - y\_pred) >= svm\_reg.epsilon)

return np.argwhere(off\_margin)

svm\_reg1.support\_ = find\_support\_vectors(svm\_reg1, X, y)

svm\_reg2.support\_ = find\_support\_vectors(svm\_reg2, X, y)

eps\_x1 = 1

eps\_y\_pred = svm\_reg1.predict([[eps\_x1]])

**[9]** def plot\_svm\_regression(svm\_reg, X, y, axes):

x1s = np.linspace(axes[0], axes[1], 100).reshape(100, 1)

y\_pred = svm\_reg.predict(x1s)

plt.plot(x1s, y\_pred, "k-", linewidth=2, label=r"$\hat{y}$")

plt.plot(x1s, y\_pred + svm\_reg.epsilon, "k--")

plt.plot(x1s, y\_pred - svm\_reg.epsilon, "k--")

plt.scatter(X[svm\_reg.support\_], y[svm\_reg.support\_], s=180, facecolors='#FFAAAA')

plt.plot(X, y, "bo")

plt.xlabel(r"$x\_1$", fontsize=18)

plt.legend(loc="upper left", fontsize=18)

plt.axis(axes)

fig, axes = plt.subplots(ncols=2, figsize=(9, 4), sharey=True)

plt.sca(axes[0])

plot\_svm\_regression(svm\_reg1, X, y, [0, 2, 3, 11])

plt.title(r"$\epsilon = {}$".format(svm\_reg1.epsilon), fontsize=18)

plt.ylabel(r"$y$", fontsize=18, rotation=0)

#plt.plot([eps\_x1, eps\_x1], [eps\_y\_pred, eps\_y\_pred - svm\_reg1.epsilon], "k-", linewidth=2)

plt.annotate(

'', xy=(eps\_x1, eps\_y\_pred), xycoords='data',

xytext=(eps\_x1, eps\_y\_pred - svm\_reg1.epsilon),

textcoords='data', arrowprops={'arrowstyle': '<->', 'linewidth': 1.5}

)

plt.text(0.91, 5.6, r"$\epsilon$", fontsize=20)

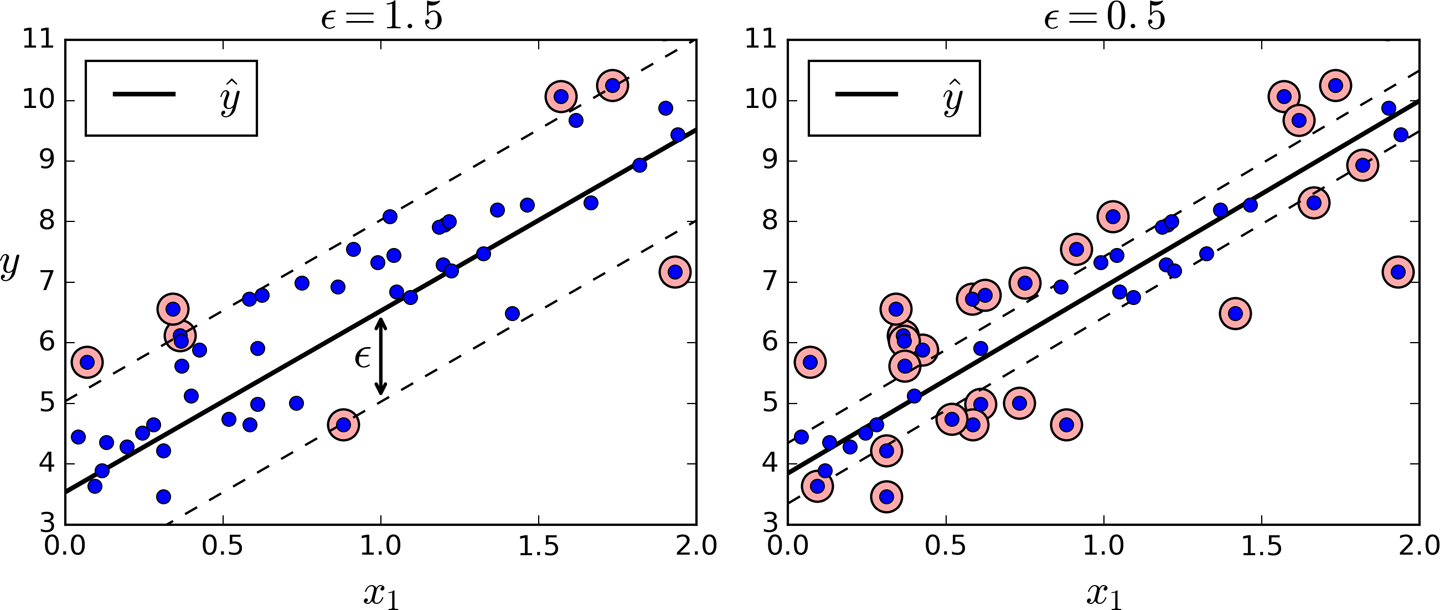
plt.sca(axes[1])

plot\_svm\_regression(svm\_reg2, X, y, [0, 2, 3, 11])

plt.title(r"$\epsilon = {}$".format(svm\_reg2.epsilon), fontsize=18)

save\_fig("svm\_regression\_plot")

plt.show()



**[10]** np.random.seed(42)

m = 100

X = 2 \* np.random.rand(m, 1) - 1

y = (0.2 + 0.1 \* X + 0.5 \* X\*\*2 + np.random.randn(m, 1)/10).ravel()

**[11]** from sklearn.svm import SVR

svm\_poly\_reg = SVR(kernel="poly", degree=2, C=100, epsilon=0.1, gamma="scale")

svm\_poly\_reg.fit(X, y)

**[12]** from sklearn.svm import SVR

svm\_poly\_reg1 = SVR(kernel="poly", degree=2, C=100, epsilon=0.1, gamma="scale")

svm\_poly\_reg2 = SVR(kernel="poly", degree=2, C=0.01, epsilon=0.1, gamma="scale")

svm\_poly\_reg1.fit(X, y)

svm\_poly\_reg2.fit(X, y)

**[13]** fig, axes = plt.subplots(ncols=2, figsize=(9, 4), sharey=True)

plt.sca(axes[0])

plot\_svm\_regression(svm\_poly\_reg1, X, y, [-1, 1, 0, 1])

plt.title(r"$degree={}, C={}, \epsilon = {}$".format(svm\_poly\_reg1.degree, svm\_poly\_reg1.C, svm\_poly\_reg1.epsilon), fontsize=18)

plt.ylabel(r"$y$", fontsize=18, rotation=0)

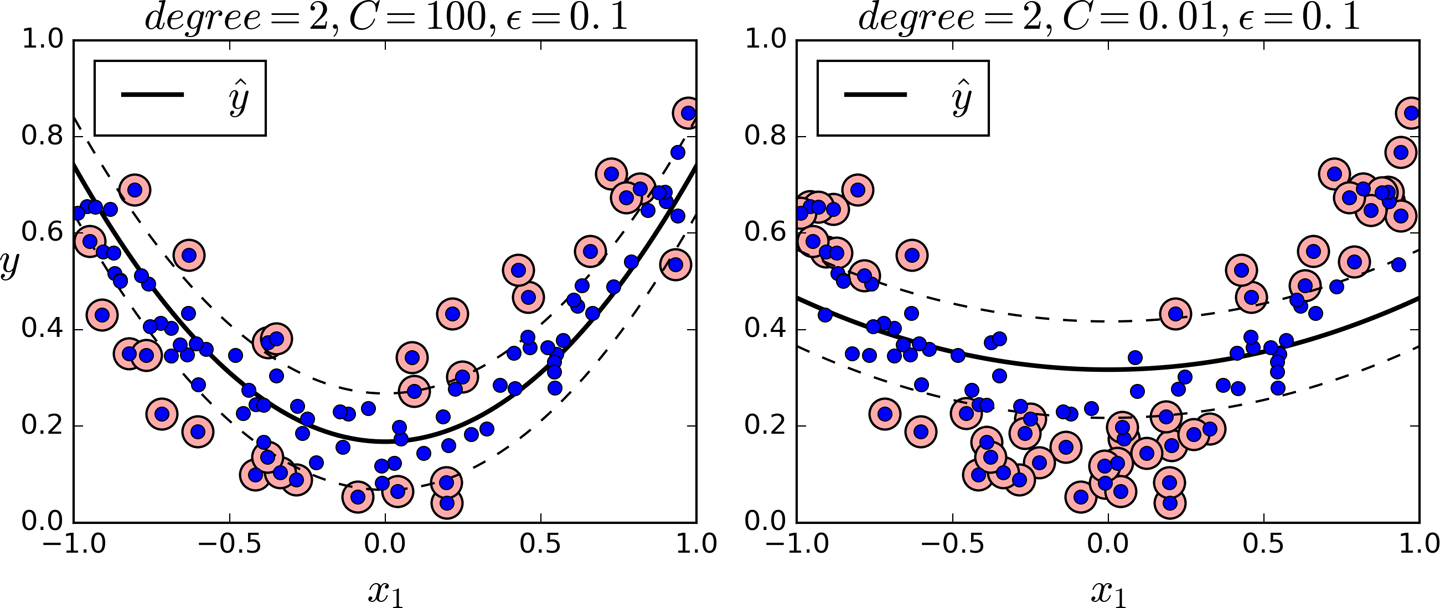
plt.sca(axes[1])

plot\_svm\_regression(svm\_poly\_reg2, X, y, [-1, 1, 0, 1])

plt.title(r"$degree={}, C={}, \epsilon = {}$".format(svm\_poly\_reg2.degree, svm\_poly\_reg2.C, svm\_poly\_reg2.epsilon), fontsize=18)

save\_fig("svm\_with\_polynomial\_kernel\_plot")

plt.show()



**[14]** t = np.linspace(-2, 4, 200)

h = np.where(1 - t < 0, 0, 1 - t) # max(0, 1-t)

plt.figure(figsize=(5,2.8))

plt.plot(t, h, "b-", linewidth=2, label="$max(0, 1 - t)$")

plt.grid(True, which='both')

plt.axhline(y=0, color='k')

plt.axvline(x=0, color='k')

plt.yticks(np.arange(-1, 2.5, 1))

plt.xlabel("$t$", fontsize=16)

plt.axis([-2, 4, -1, 2.5])

plt.legend(loc="upper right", fontsize=16)

save\_fig("hinge\_plot")

plt.show()

