LINEAR SVM

Due Date : 9/28 Monday 10:15 PM EST

In [1]:
```
pip install libsvm
```

```
Requirement already satisfied: libsvm in c:\users\kaike\anaconda3\lib\site-pa
ckages (3.23.0.4)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.io as io
import libsvm
import math
from sklearn.svm import SVC
from libsvm.svmutil import *

%matplotlib inline
```

3.1 Linear Support Vector Machine on toy data

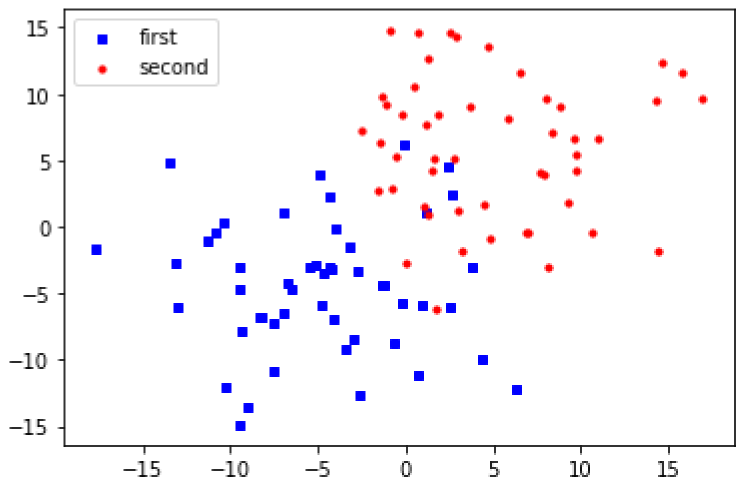
3.1.1

Generate a training set of size 100 with 2D features (X) drawn at random as follows:

- X_{neg} $\sim\mathcal{N}$([-5, -5], 5*I_2) and correspond to negative labels (-1)
- X_{pos} $\sim\mathcal{N}$([5, 5], 5*I_2) and correspond to positive labels (+1)
  Accordingly, $X=[X_{neg}, X_{pos}]$ is a 100×2 array, Y is a 100×1 array of values $\in\{-1,1\}$.

  Draw a scatter plot of the full training dataset with the points colored according to their labels.

In [3]:
```
import random



# Generate binary class dataset
np.random.seed(0)

n_samples = 100
center_1 = [-5, -5]
center_2 = [5, 5]

# random number of X is Xneg
random_seperation = 50
#int(n_samples*random.uniform(0, 1))

# Generate Data:
Y = []
Xneg = np.random.normal(-5,5,size = (int(random_seperation),2))
for i in range(int(random_seperation)):
    Y.append(-1)
Xpos = np.random.normal(5,5,size = (int(100 - random_seperation),2))
for i in range(int(100 - random_seperation)):
    Y.append(1)
X = np.concatenate((Xneg, Xpos))



# Scatter plot:
X_neg_plot_1 = []
X_neg_plot_2 = []
X_pos_plot_1 = []
X_pos_plot_2 = []
for i in range(len(X)):
    if Y[i] == -1 :
        X_neg_plot_1.append(X[i][0])
        X_neg_plot_2.append(X[i][1])
    elif Y[i] == 1 :
        X_pos_plot_1.append(X[i][0])
        X_pos_plot_2.append(X[i][1])


plt.scatter(X_neg_plot_1, X_neg_plot_2, s=10, c='b', marker="s", label='first'
)
plt.scatter(X_pos_plot_1, X_pos_plot_2, s=10, c='r', marker="o", label='secon
d')
plt.legend(loc='upper left');
plt.show()
```

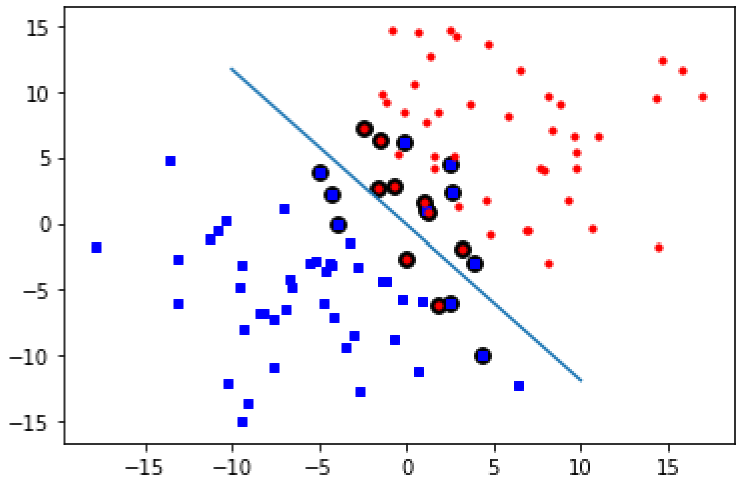


3.1.2

Train a linear support vector machine on the data with $C = 1$ and draw the decision boundary line that separates o and x. Mark the support vectors separately (ex.circle around the point).

Note: You can use the libsvm.svmutil functions with the kernel_type set to 0, indiciating a linear kernel and svm_type set to 0 indicating C-SVC. Also note that the support_vector coefficients returned by the LIBSVM model are the dual coefficients.

In [4]:
```
# Define the SVM problem
problem = svm_problem(Y,X)
# Define the hyperparameters
param = svm_parameter('-t 0 -s 0')
# Train the model
model = svm_train(problem, param)

# Compute the slope and intercept of the separating line/hyperplanee with the
 use of the support vectors
# and other information from the LIBSVM model.
sv = model.get_SV()
sv_coef = model.get_sv_coef()
w = np.matmul(np.array(X)[np.array(model.get_sv_indices()) - 1].T, sv_coef)
b = -model.rho.contents.value
if model.get_labels()[1] == -1:
    w = -w
    b = -b


# Draw the scatter plot, the decision boundary line, and mark the support vect
ors.

# Plot the line
x_bdd = [-10,10]
y_bdd = []
y_bdd.append(-(x_bdd[0] * w[0] + b) / w[1])
y_bdd.append(-(x_bdd[1] * w[0] + b) / w[1])
plt.plot(x_bdd, y_bdd)
# Plot supporting vector
for i in model.get_sv_indices():
    plt.scatter(X[i - 1][0], X[i - 1][1], color='black', s=60)
# Plot all traning point
plt.scatter(X_neg_plot_1, X_neg_plot_2, s=10, c='b', marker="s")
plt.scatter(X_pos_plot_1, X_pos_plot_2, s=10, c='r', marker="o")

plt.show()
```

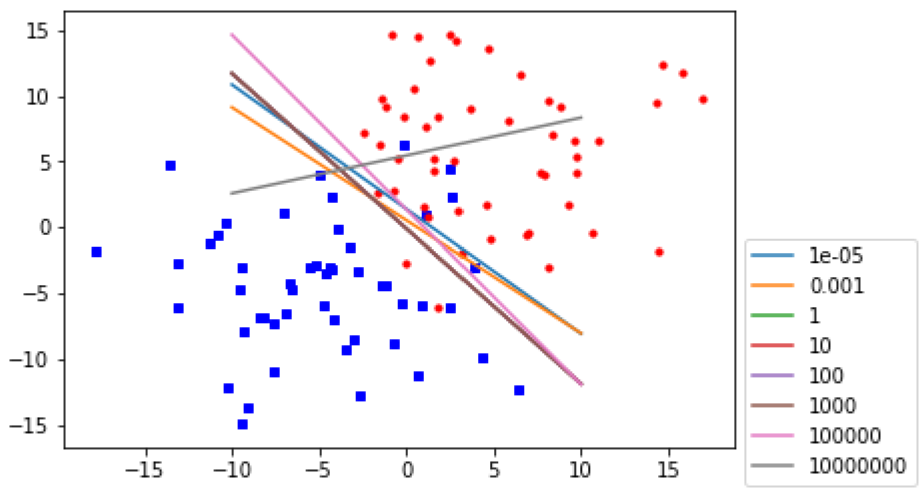


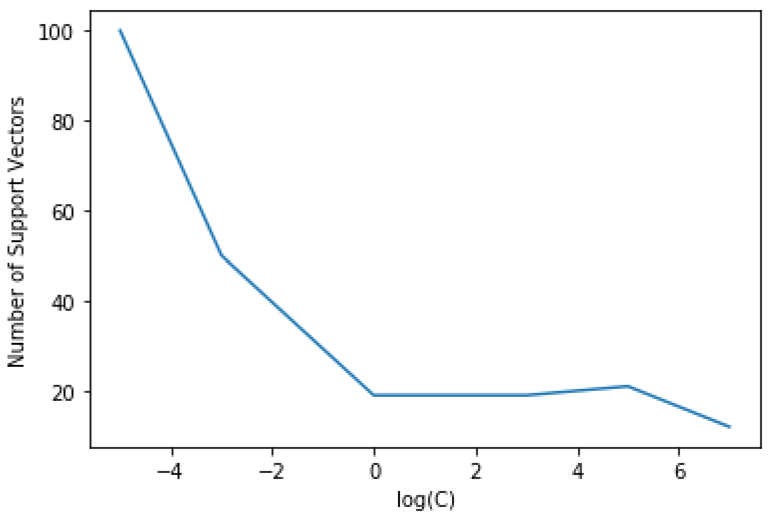
3.1.3

Draw a line that separates the data for 8 different C (10^{-5}~10^7). Plot the number of support vectors vs. C (plot x-axis on a log scale). How does the number of support vectors change as C increases and why does it change like that?

Note: You might prefer to use the command-line style of svm_parameter initialization such as: svm_parameter('-s 0 -t 0') to indicate a linear kernel and C-SVC as the SVM type.

In [5]:
```
C_range = [10**-5, 10**-3, 1, 10, 100, 10**3, 10**5, 10**7]
num_sv = []

# Loop over a similar setup to that in the previous code block.
for C in C_range:
    param = svm_parameter('-s 0 -t 0 -c ' + str(C))
    model = svm_train(problem, param)
    num_sv.append(model.get_nr_sv())
    sv = model.get_SV()
    sv_coef = model.get_sv_coef()

    # Calculate w and b
    w = np.matmul(np.array(X)[np.array(model.get_sv_indices()) - 1].T, sv_coef
)
    b = -model.rho.contents.value
    if model.get_labels()[1] == -1:
        w = -w
        b = -b

    # Draw the scatter plot with multiple decision lines on top (one for each
 value of C)
    x_bdd = [-10,10]
    y_bdd = []
    y_bdd.append(-(x_bdd[0] * w[0] + b) / w[1])
    y_bdd.append(-(x_bdd[1] * w[0] + b) / w[1])
    plt.plot(x_bdd, y_bdd, label = str(C))


# Plot all traning point
plt.scatter(X_neg_plot_1, X_neg_plot_2, s=10, c='b', marker="s")
plt.scatter(X_pos_plot_1, X_pos_plot_2, s=10, c='r', marker="o")
plt.legend(loc='upper left', bbox_to_anchor=(1, 0.5))
plt.show()

# Draw the num_sv vs. C plot
log_C = []
for C in C_range:
    log_C.append(math.log(C,10))
plt.xlabel('log(C)')
plt.ylabel('Number of Support Vectors')
plt.plot(log_C, num_sv)
plt.show()
```





> How does the number of support vectors change as C increases and why does it change like that?

C governs the importance of avoiding misclassifying each training sample. A high C suggests that the SVM penalizes the misclasification very badly. Support vectors are traning sample which the output of SVM has a value between [0,1]. If I increase C, the margin will be narrower because the main goal is to reduce misclassification. Therefore, less vectors will be support vectors.

3.1.4

Now try rescaling the data to the [0,1] range and repeat the steps of the previous question (3.1.3) and over the same range of C values. Are the decision boundaries different from those in the previous question? What does this imply about (a) the geometric margin and (b) the relative effect of each feature on the predictions of the trained model ?

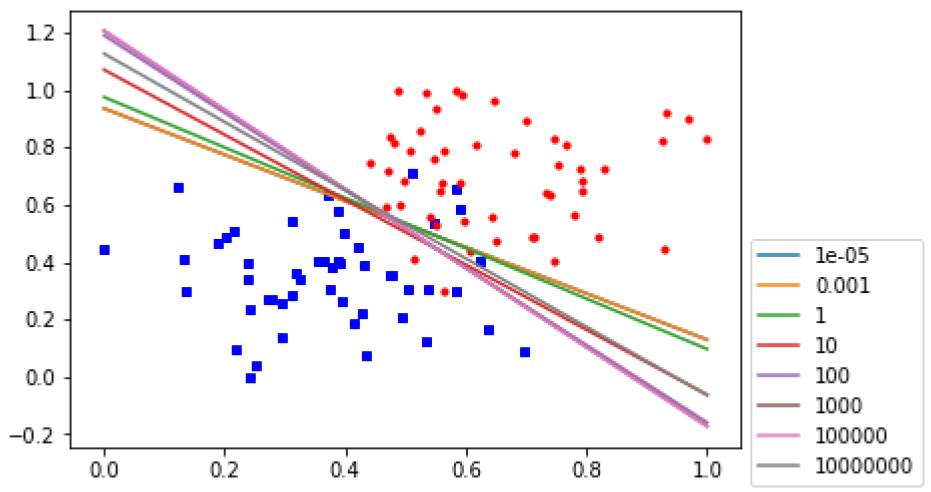
Solution below:

SVM tries to maximize the distance between the separating plane and the support vectors. If one feature (i.e. one dimension in this space) has very large values, it will dominate the other features when calculating the distance. If you rescale all features (e.g. to [0, 1]), they all have the same influence on the distance metric.

In this case, the decision boundary will shift and shirk the same way with the changes of features. But the change is propotional, because the we rescale two features by very similar proportion (max(feature1) - min(feature1) is approx equal to max(feature2) - min(feature2)), the geometric margin will decrease proportionally and the relative effect of each feature remains pretty much unchanged. However if we rescale each feature differently, the relative effect of each feature will be changed.

In [6]:
```
import sklearn
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler()

# Single line below:
X_train_minmax = min_max_scaler.fit_transform(X)
```

In [7]:
```
C_range = [10**-5, 10**-3, 1, 10, 100, 10**3, 10**5, 10**7]
num_sv = []

# Repeat the loop from 3.1.3
problem = svm_problem(Y,X_train_minmax)
for C in C_range:
    param = svm_parameter('-s 0 -t 0 -c ' + str(C))
    model = svm_train(problem, param)
    num_sv.append(model.get_nr_sv())
    sv = model.get_SV()
    sv_coef = model.get_sv_coef()

    # Calculate w and b
    w = np.matmul(np.array(X_train_minmax)[np.array(model.get_sv_indices()) -
1].T, sv_coef)
    b = -model.rho.contents.value
    if model.get_labels()[1] == -1:
        w = -w
        b = -b

    # Draw the scatter plot with multiple decision lines on top (one for each
 value of C)
    x_bdd = [0,1]
    y_bdd = []
    y_bdd.append(-(x_bdd[0] * w[0] + b) / w[1])
    y_bdd.append(-(x_bdd[1] * w[0] + b) / w[1])
    plt.plot(x_bdd, y_bdd, label = str(C))


# Plot all traning point
X_neg_plot_1_minmax = []
X_neg_plot_2_minmax = []
X_pos_plot_1_minmax = []
X_pos_plot_2_minmax = []
for i in range(len(X)):
    if Y[i] == -1 :
        X_neg_plot_1_minmax.append(X_train_minmax[i][0])
        X_neg_plot_2_minmax.append(X_train_minmax[i][1])
    elif Y[i] == 1 :
        X_pos_plot_1_minmax.append(X_train_minmax[i][0])
        X_pos_plot_2_minmax.append(X_train_minmax[i][1])
plt.scatter(X_neg_plot_1_minmax, X_neg_plot_2_minmax, s=10, c='b', marker="s")
plt.scatter(X_pos_plot_1_minmax, X_pos_plot_2_minmax, s=10, c='r', marker="o")
plt.legend(loc='upper left', bbox_to_anchor=(1, 0.5))
plt.show()
```



3.2 MNIST

Multiclass kernel SVM. In this problem, we'll use support vector machines to classify the MNIST data set of handwritten digits.

3.2.1

Load in the MNIST data using from the provided mnist-original.mat file on sakai. First split the data into training and testing by simply taking the first 60k points as training and the rest as testing. Then sample 500 data points for each of the 10 categories (for a total of 5000 training points) from the 60k training photos. These 5k points are now our training set. Finally, sample 500 data points for each of the 10 categories from the 10k testing photos. These 5k points are now our testing set.

Note: For data loading, you might want to use scipy.io.loadmat.

In [8]:
```
import scipy
import scipy.io
from scipy.io import loadmat

np.random.seed(0)

minist_data = loadmat('mnist-original.mat')
X = minist_data['data'].transpose()
y = minist_data['label'].reshape(-1)

X_train_pool = X[:60000,:]
y_train_pool = y[:60000]
X_test_pool = X[60000:,:]
y_test_pool = y[60000:]

# get the index for sample selection
index = []

for label in set(y_train_pool):
    index_temp = []
    index_temp = np.argwhere(y_train_pool == label).reshape(-1)
    index += list(np.random.choice(index_temp, size = 500, replace = False))

X_train = np.take(X_train_pool, index, axis = 0)
y_train = np.take(y_train_pool, index, axis = 0)

index = []

for label in set(y_test_pool):
    index_temp = []
    index_temp = np.argwhere(y_test_pool == label).reshape(-1)
    index += list(np.random.choice(index_temp, size = 500, replace = False))

X_test = np.take(X_test_pool, index, axis = 0)
y_test = np.take(y_test_pool, index, axis = 0)

#print(X_train.shape)
#print(y_train.shape)
```

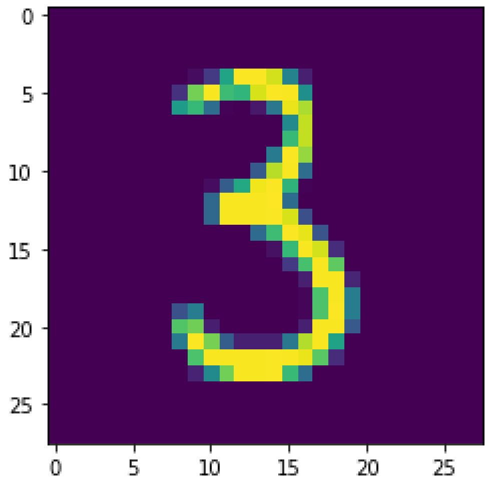
In [9]:
```
np.unique(y_train, return_counts=True) #ensure each label has 500 examples.
```

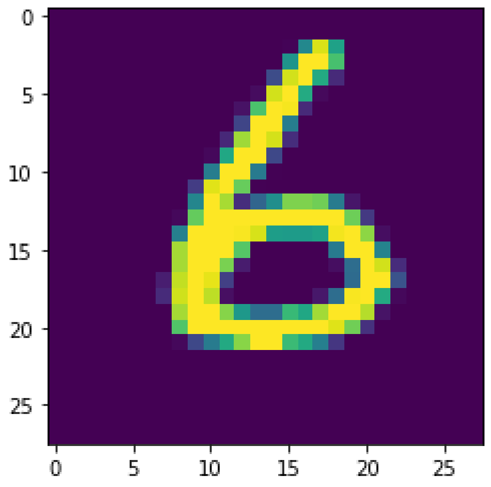
Out[9]:
```
(array([0., 1., 2., 3., 4., 5., 6., 7., 8., 9.]),
 array([500, 500, 500, 500, 500, 500, 500, 500, 500, 500], dtype=int64))
```

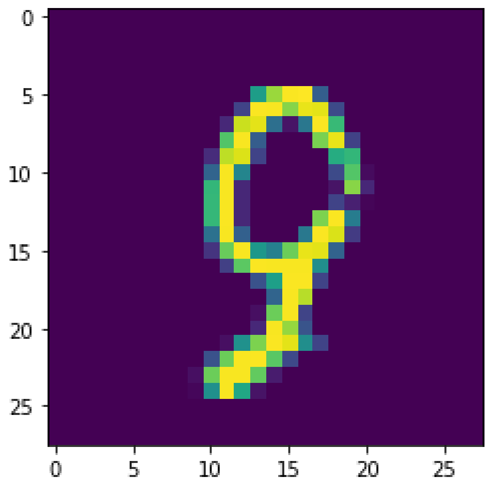
3.2.2

Draw 3 different digits using pyplot.imshow().

In [10]:
```python
from random import randrange

digits = [3,6,9]
for digit in digits:
    index_temp = []
    index_temp = np.argwhere(y_train == digit).reshape(-1)
    plt.imshow(X_train[index_temp[randrange(500)],:].reshape(28,28))
    plt.show()
```







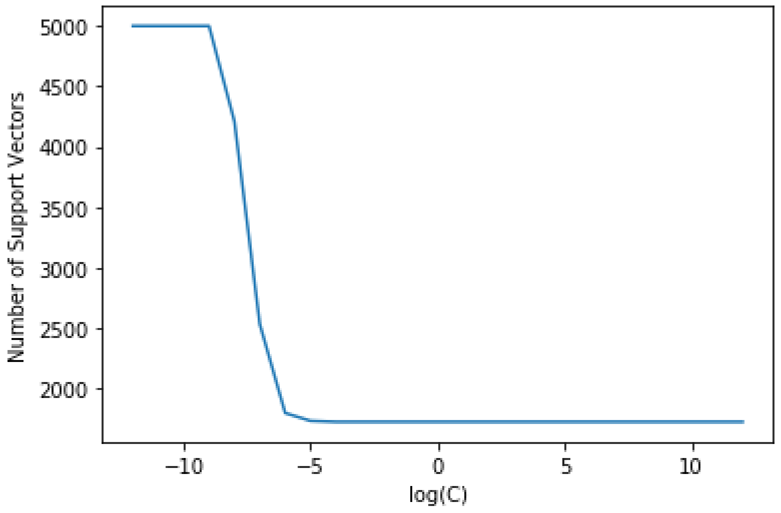
3.2.3

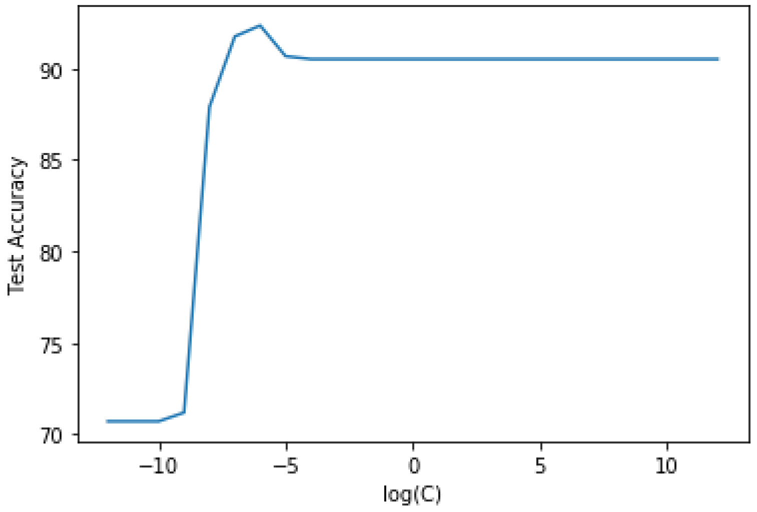
For each value $C = 10^{-12}$~10^{12} train a support vector machine with a linear kernel and compute its accuracy on the test set subsampled previously. Plot test accuracy and the number of support vectors (two separate plots) vs. C for $C = 10^{-12}$~10^{12} (plot 7 points or more with the x-axis on a log scale).

In [11]:
```
C_range = []
for i in range(-12, 13):
    C_range.append(10**i)

accuracy = []
num_sv = []
for C in C_range:
    # Define the SVM problem
    problem = svm_problem(y_train, X_train)
    # Define the hyperparameters
    param = svm_parameter('-s 0 -t 0 -c ' + str(C))
    # Train the model
    model = svm_train(problem, param)
    # make prediction
    p_label, p_acc, p_val = svm_predict(y_test, X_test, model)
    accuracy.append(p_acc[0])
    # number of sv
    num_sv.append(model.get_nr_sv())
```

```
Accuracy = 70.72% (3536/5000) (classification)
Accuracy = 70.72% (3536/5000) (classification)
Accuracy = 70.72% (3536/5000) (classification)
Accuracy = 71.2% (3560/5000) (classification)
Accuracy = 87.9% (4395/5000) (classification)
Accuracy = 91.76% (4588/5000) (classification)
Accuracy = 92.36% (4618/5000) (classification)
Accuracy = 90.68% (4534/5000) (classification)
Accuracy = 90.52% (4526/5000) (classification)
Accuracy = 90.52% (4526/5000) (classification)
Accuracy = 90.52% (4526/5000) (classification)
Accuracy = 90.52% (4526/5000) (classification)
Accuracy = 90.52% (4526/5000) (classification)
Accuracy = 90.52% (4526/5000) (classification)
Accuracy = 90.52% (4526/5000) (classification)
Accuracy = 90.52% (4526/5000) (classification)
Accuracy = 90.52% (4526/5000) (classification)
Accuracy = 90.52% (4526/5000) (classification)
Accuracy = 90.52% (4526/5000) (classification)
Accuracy = 90.52% (4526/5000) (classification)
Accuracy = 90.52% (4526/5000) (classification)
Accuracy = 90.52% (4526/5000) (classification)
Accuracy = 90.52% (4526/5000) (classification)
Accuracy = 90.52% (4526/5000) (classification)
Accuracy = 90.52% (4526/5000) (classification)
```

In [12]:
```
log_C = []
for C in C_range:
    log_C.append(math.log(C,10))

plt.plot(log_C, num_sv)
plt.xlabel('log(C)')
plt.ylabel('Number of Support Vectors')
plt.show()

plt.plot(log_C, accuracy)
plt.xlabel('log(C)')
plt.ylabel('Test Accuracy')
plt.show()
```





In [ ]: