Problem 1

d is the correct answer.
We have this because we can vary $\mathbf{w}$ and b. And since $\mathbf{w}$ has d variables, we have $d+1$ variables in total.

Problem 2

a is the correct answer.
I wrote some code to implement SVM with soft margin on the zip-code data by solving the equations laid out in the problem set. I used binary classification error. To help me write this code, I used the libsvm library. For this problem, I made sure to use the polynomial kernel as given in the problem. After running this code for 0 versus all, 2 versus all, 4 versus all, 6 versus all, and 8 versus all, I found that 0 versus all gave me the lowest accurary and thus the highest E_{in} (ran prediction code on the training data).

Problem 3

a is the correct answer.
I wrote some code to implement SVM with soft margin on the zip-code data by solving the equations laid out in the problem set. I used binary classification error. To help me write this code, I used the libsvm library. For this problem, I made sure to use the polynomial kernel as given in the problem. After running this code for 1 versus all, 3 versus all, 5 versus all, 6 versus all, and 9 versus all, I found that 1 versus all gave me the highest accurary and thus the lowest E_{in} (ran prediction code on the training data).

Problem 4

c is the correct answer.
Using the code I wrote for the above two problems, I got 386 support vectors for 1 versus all and 2180 support vectors for 0 versus all (with all the other parameters as specified in the problem). The difference between these two numbers is around 1800.

Problem 5

d is the correct answer.
I wrote some code that runs the 1 versus 5 classifier with $Q = 2$ and $C \in \{0.001, 0.01, 0.1, 1\}$. I then observed the number of support vectors, E_{in}, and E_{out} for each C value. I found that the only statement given in the problem that is true is that the maximum C (1) achieves the lowest E_{in} and the highest accuracy (around 99.68% accuracy). I found that all the other statements given in the problem are false.

Problem 6

b is the correct answer.
I wrote some code that runs the 1 versus 5 classifier with $Q = 2$ and $C \in \{0.0001, 0.001, 0.01, 0.1, 1\}$ and then runs the 1 versus 5 classifier with $Q = 5$ and $C \in \{0.0001, 0.001, 0.01, 0.1, 1\}$. I did this so I could compare values between all the runs. I found that the only statement given in the problem that is true is that when $C = 0.001$, the number of support vectors is lower at $Q = 5$ (76 versus 25).

Problem 7

b is the correct answer.
I wrote some code that runs 10-fold cross validation for the polynomial kernel. With this code, I considered the 1 versus 5 classifier with $Q = 2$. I used E_{cv} to select $C \in \{0.0001, 0.001, 0.01, 0.1, 1\}$. If there was a tie in E_{cv}, I selected the smaller C. I then ran this code for 100 random runs (so I tried 100 different partitions). With this, I discovered that $C = 0.001$ is selected most often.

Problem 8

c is the correct answer.
I basically used the same code I wrote for the above problem, and for each run, I added the value of E_{cv} for $C = 0.001$. I then took the average value of this list, and it was around 0.005.

Problem 9

e is the correct answer.
I wrote code that runs SVM with the RBF kernel for all the values of C given in the problem. I ran this code on the 1 versus 5 classifier. To help me write this code, I used the libsvm library. The value $C = 10^6$ gave me the lowest E_{in} value (ran prediction code on the training data).

Problem 10

c is the correct answer.
I wrote code that runs SVM with the RBF kernel for all the values of C given in the problem. I ran this code on the 1 versus 5 classifier. To help me write this code, I used the libsvm library. The value $C = 100$ gave me the lowest E_{out} value (ran prediction code on the test data).

Code

```
from __future__ import division
from svm import *
from svmutil import *
from collections import Counter
import sys
import random
import numpy as np

class MySvm:
    def __init__(self, file_train, file_test):
        self.training_labels = self.load_labels(file_train)
        self.training_data = self.load_data(file_train)
        self.training_data_curr = self.training_data
        self.scores_tr = []

        self.test_labels = self.load_labels(file_test)
        self.test_data = self.load_data(file_test)
        self.test_data_curr = self.test_data
        self.scores_test = []

        self.cross_data_curr_list = []
        self.scores_cross_list = []
        self.scores_tr_list = []
        self.training_data_curr_list = []

        self.model = None
        self.model_list = []
        self.num_support_vectors = 0

    def load_labels(self, file_name):
        file_obj = open(file_name)
        labels = []
        for line in file_obj:
            labels.append(float(line.split()[0]))
        return labels

    def load_data(self, file_name):
        file_obj = open(file_name)
        data = []
        for line in file_obj:
            item = []
            line_split = line.split()
            for i in range(1, len(line_split)):
                item.append(float(line_split[i]))
            data.append(item)
        return data

    def one_versus_all(self, num_one, kernel_type, error_const, poly_degree,
            is_cross, is_repeat = False):
        if not is_repeat:
            self.scores_tr, self.training_data_curr = self.get_one_versus_all_lists(
                    num_one, self.training_labels, self.training_data)
            self.scores_test, self.test_data_curr = self.get_one_versus_all_lists(
                    num_one, self.test_labels, self.test_data)
            if is_cross:
                self.scores_tr, self.training_data_curr, self.scores_cross, self.cross_data

        prob = svm_problem(self.scores_tr, self.training_data_curr)
        param_str = ’-t %d -r 1 -g 1 -c %f -d %d’ %(kernel_type, error_const, poly_degree)
        param = svm_parameter(param_str)
        model = svm_train(prob, param)
        self.model = model
        self.num_support_vectors = len(model.get_SV())

    def one_versus_one(self, num_one, num_other, kernel_type, error_const,
            poly_degree, is_cross, is_repeat = False):
        if not is_repeat:
            self.scores_tr, self.training_data_curr = self.get_one_versus_one_lists(
                    num_one, num_other, self.training_labels, self.training_data)
            self.scores_test, self.test_data_curr = self.get_one_versus_one_lists(
                    num_one, num_other, self.test_labels, self.test_data)
            if is_cross:
                self.scores_tr_list, self.training_data_curr_list, self.scores_cross_list,

        if not is_cross:
            prob = svm_problem(self.scores_tr, self.training_data_curr)
            param_str = '-t %d -r 1 -g 1 -c %f -d %d' %(kernel_type, error_const, poly_degr
            param = svm_parameter(param_str)
            model = svm_train(prob, param)
            self.model = model
            self.num_support_vectors = len(model.get_SV())
        else:
            self.model_list = []
            sum_num_support_vectors = 0
            for i in range(0, len(self.scores_tr_list)):
                prob = svm_problem(self.scores_tr_list[i], self.training_data_curr_list[i])
                param_str = '-t %d -r 1 -g 1 -c %f -d %d' %(kernel_type, error_const, poly_
                param = svm_parameter(param_str)
                model = svm_train(prob, param)
                self.model_list.append(model)
                sum_num_support_vectors += len(model.get_SV())
            self.num_support_vectors = sum_num_support_vectors / len(self.scores_tr_list)

    def get_one_versus_all_lists(self, num_one, labels, data_items):
        scores = []
        for label in labels:
            if label == num_one:
                scores.append(1)
            else:
                scores.append(-1)
        return (scores, data_items)

    def get_one_versus_one_lists(self, num_one, num_other, labels, data_items):
        scores = []
        data_curr = []
        for i in range(0, len(labels)):
            label = labels[i]
            item = data_items[i]
            if label == num_one:
                scores.append(1)
                data_curr.append(item)
            elif label == num_other:
                scores.append(-1)
                data_curr.append(item)
        return (scores, data_curr)

    def get_cross_val_lists(self):
        z_list = list(zip(self.scores_tr, self.training_data_curr))
        random.shuffle(z_list)
        shuffled_scores_tr, shuffled_training_data_curr = zip(*z_list)
        index = int(len(self.scores_tr) / 10)

        scores_tr_list, training_data_curr_list, scores_cross_list, cross_data_curr_list =
        start = 0
        end = index
        while end < len(self.scores_tr):
            if end + index >= len(self.scores_tr):
                end = len(self.scores_tr)
            scores_cross_list.append(list(shuffled_scores_tr[start:end]))
            cross_data_curr_list.append(list(shuffled_training_data_curr[start:end]))
            scores_tr_list.append(list(shuffled_scores_tr[0:start]) + list(shuffled_scores_
            training_data_curr_list.append(list(shuffled_training_data_curr[0:start]) + lis
            start += index
            end += index
        return (scores_tr_list, training_data_curr_list, scores_cross_list, cross_data_curr

    def get_error_in(self):
        p_labels, p_acc, p_vals = svm_predict(self.scores_tr,
                self.training_data_curr, self.model)
        return p_acc[0]

    def get_error_out(self):
        p_labels, p_acc, p_vals = svm_predict(self.scores_test,
                self.test_data_curr, self.model)
        return p_acc[0]

    def get_error_cv(self):
        sum_acc = 0
        for i in range(0, len(self.scores_cross_list)):
            p_labels, p_acc, p_vals = svm_predict(self.scores_cross_list[i],
                    self.cross_data_curr_list[i], self.model_list[i])
            sum_acc += p_acc[0]
        print '========LENGTH = ', len(self.scores_cross_list)
        return sum_acc / len(self.scores_cross_list)

if __name__ == '__main__':
    prob1_3 = False
    prob5 = False
    prob6 = False
    prob7_8 = True
    prob9_10 = False
    print 'Kshit is gay'
    my_svm = MySvm('features.train', 'features.test')

    if prob1_3:
        for i in range(1, 11, 2):
            print 'i = ', i
            my_svm.one_versus_all(i, 1, .01, 2, False)
            p_acc = my_svm.get_error_in()
            print 'Num support vectors = ', my_svm.num_support_vectors
            print '\n\n'

    if prob5:
        print 'Q = ', 2
        for c in [.0001, .001, .01, .1, 1]:
            print 'C = ', c
            my_svm.one_versus_one(1, 5, 1, c, 2, False)
            print 'E in'
            my_svm.get_error_in()
            print 'E out'
            my_svm.get_error_out()
            print 'Num support vectors = ', my_svm.num_support_vectors
            print '\n'
        print '\n\n\n'

    if prob6:
        print 'Q = ', 5
        for c in [.0001, .001, .01, .1, 1]:
            print 'C = ', c
            my_svm.one_versus_one(1, 5, 1, c, 5, False)
            print 'E in'
            my_svm.get_error_in()
            print 'E out'
            my_svm.get_error_out()
            print 'Num support vectors = ', my_svm.num_support_vectors
            print '\n'

    if prob7_8:
        c_list = []
        e_dict = {.0001 : [], .001 : [], .01 : [], .1 : [], 1 : []}
        for i in range(0, 100):
            min_error = sys.maxint
            c_val = 1
            repeat = False
            for c in [.0001, .001, .01, .1, 1]:
                my_svm.one_versus_one(1, 5, 1, c, 2, True, repeat)
                repeat = True
                # Get error, NOT accurary
                error = (100 - my_svm.get_error_cv()) / 100
                e_dict[c].append(error)
                if error < min_error:
                    min_error = error
                    c_val = c
            c_list.append(c_val)
        c_data = Counter(c_list)
        print 'Counts = ', c_data.most_common()
        print 'Mode = ', c_data.most_common(1)
        for key, val in e_dict.iteritems():
            new_val = []
            val_max = max(val)
            val_min = min(val)
            val_mean = sum(val) / len(val)
            new_val = [val_mean, val_max, val_min]
            e_dict[key] = new_val
        print e_dict

    if prob9_10:
        min_error_in = sys.maxint
        min_error_out = sys.maxint
        c_val_in = 0
        c_val_out = 0
        for c in [.01, 1, 100, 10000, 1000000]:
            my_svm.one_versus_one(1, 5, 2, c, 2, False, False)
            error_in = (100 - my_svm.get_error_in()) / 100
            if error_in < min_error_in:
                min_error_in = error_in
                c_val_in = c
            error_out = (100 - my_svm.get_error_out()) / 100
            if error_out < min_error_out:
                min_error_out = error_out
                c_val_out = c
        print 'Min error in = ', min_error_in
        print 'C value in = ', c_val_in
        print 'Min error out = ', min_error_out
        print 'C value out = ', c_val_out
```