HW 8

CS156, Kai Chang

Question 1

Answer Choice: **D**, a quadratic programming problem with $d + 1$ variables
Reasoning: Given d represents the diemsnionality of the space, we know that for the primal problem we want to minimize $\frac{1}{2} w^T w$ such that for all $i, y_i(w^T x_i + w_0) \geq 1$.

And because the weight is the variable we are minimizing, thus this can be solved using a quadratic program with $d + 1$ variables, and an i constraints. Note that it is $d + 1$ instead of d because we also have to consider the w_0 term (ie. the bias). Also note for the dual problem, we look at the size of the data set N, rather than the dimensionality of the space.

Question 2

Answer Choice: **A**, 0 versus all
Reasoning: see code. Note we are using E_in, not E_out.

In [17]:
```
'''
Used packages from: http://www.csie.ntu.edu.tw/~cjlin/libsvm/
Data is structured in text file as : digit intensity symmetry
Data will be structured in program as: [[digit, intensity, symmetry]]
'''

import numpy as np

t_d = np.loadtxt('features.train')
x_train = t_d[:,1:]
y_train = t_d[:,0]
```

In [36]:
```
def poly_kernel(x_n, x_m, Q):
    '''
    Polynomial kernal used in the soft margin SVM. Format expanded looks like thi
s
    K(x, x') = (1 + x^T . x')^Q
             = (1 + x1x'1 + x2x'2 + ... + xdx'd)^Q
    where d represents the number of features per data point

    Parameters
    ----------
    x_n : first data point, numpy array
    x_m : second data point, numpy array
    Q   : degree of polynomial, int

    Returns
    -------
    equivalent kernel
    '''

    return (1.0 + np.dot(x_n.T, x_m))**Q
```

In [37]:
```
def bin_class(y, n):
    '''
    Applies the one-versus-all binary classifier. Can be used for the
    one-versus-one binary classifier through a computational trick.

    Parameters
    ----------
    y : dataset output, numpy array
    n : one digit value to classify as +1

    Returns
    -------
    numpy array of binary classified values
    '''
    y_bin = -np.ones(len(y))
    y_bin[y == n] = 1
    return y_bin
```

In [48]:
```
from sklearn import svm

'''
The kernel function can be any of the following:
    linear: <x, x'>
    polynomial: (γ<x, x'> + r)^d, γ = gamma, d = degree, r = coef0
    rbf: exp(-γ|x-x'|^2). γ = gamma, must be > 0
    sigmoid (tanh(γ<x,x'> + r)), γ = gamma, r = coef0

From directly the link: http://scikit-learn.org/stable/modules/svm.html.
Note < > suggests dot product with transpose, ie. <x, x'> = x^T . x'
'''
# parameters for kernel, SVM
C = 0.01
Q = 2

# svm, with desired properties (attributes)
clf = svm.SVC(C=C, kernel='poly', degree=Q, gamma=1, coef0=1)

for n in [0, 2, 4, 6, 8]:
    # apply binary classification
    y_t_b = bin_class(y_train, n)
    clf.fit(x_train, y_t_b)
    y_t_b_ = clf.predict(x_train)
    E_in = np.sum(abs(y_t_b_ + y_t_b) == 0) / float(len(y_t_)) # numpy sum has bo
olean test implementation
    print 'E_in for %i versus all: ' %n, E_in
```

```
E_in for 0 versus all:  0.105883966534
E_in for 2 versus all:  0.100260595254
E_in for 4 versus all:  0.0894253188863
E_in for 6 versus all:  0.0910711836511
E_in for 8 versus all:  0.0743382252092
```

Question 3

Answer Choice: **A**, 1 versus all
Reasoning: see code result.

In [49]:
```
# parameters for kernel, SVM
C = 0.01
Q = 2

# svm, with desired properties (attributes)
clf = svm.SVC(C=C, kernel='poly', degree=Q, gamma=1, coef0=1)

for n in [1, 3, 5, 7, 9]:
    # apply binary classification
    y_t_b = bin_class(y_train, n)
    clf.fit(x_train, y_t_b)
    y_t_b_ = clf.predict(x_train)
    E_in = np.sum(abs(y_t_b_ + y_t_b) == 0) / float(len(y_t_)) # numpy sum has bo
olean test implementation
    print 'E_in for %i versus all: ' %n, E_in
```

```
E_in for 1 versus all:  0.0144013166918
E_in for 3 versus all:  0.0902482512687
E_in for 5 versus all:  0.0762584007681
E_in for 7 versus all:  0.0884652311068
E_in for 9 versus all:  0.0883280757098
```

Question 4

Answer Choice: **C**, 1800
Reasoning: see code.

In [135]:
```
# parameters for kernel, SVM
C = 0.01
Q = 2

# svm, with desired properties (attributes)
clf = svm.SVC(C=C, kernel='poly', degree=Q, gamma=1, coef0=1)

for n in [0, 1]:
    # apply binary classification
    y_t_b_temp = bin_class(y_train, n)
    clf.fit(x_train, y_t_b_temp)
    y_t_b_temp_ = clf.predict(x_train)
    E_in = np.sum(abs(y_t_b_temp + y_t_b_temp_) == 0) / float(len(y_t_b_temp_)) #
 numpy sum has boolean test implementation
    print 'Average support classifer vectors for %i versus all case: ' %n, len(cl
f.support_vectors_)
```

```
Average support classifer vectors for 0 versus all case:  2179
Average support classifer vectors for 1 versus all case:  386
```

In [136]:
```
print 'difference between the number of support vectors of these two classifiers:
 ', 2179 - 386
```

```
difference between the number of support vectors of these two classifiers:  1793
```

Question 5

Answer Choice: **D**, Maximum C achieves the lowest E_{in}
Reasoning: see code. This is using the one-versus-one classification! I originally was going to select the data using y_train == 1, and then y_train == 5 and join the two, but numpy has a neat function that handles this case. Numpy is truly powerful. We also should realize that because the other data points do not matter, we remove them from your SVM learning process. If we do not remove them, we will note that the learning will suck.

We see from the result that E_{in} does not strictly decrease, E_{out} does not strictly increase, so *C* is invalid. We also see that the support vector decreases, but not strictly, so *A* and *B* are ruled out. Now, we do see that for $C = 1$ or the maximum C in this C-set, it has the lowest E_{in}. Thus, *D* is our solution.

In [64]:
```
# load test dataset
t_d = np.loadtxt('features.test')
x_test = t_d[:,1:]
y_test = t_d[:,0]

# parameters for kernel, SVM
C_list = [0.001, 0.01, 0.1, 1]
Q = 2

x_train_ = x_train[np.logical_or(y_train==1, y_train==5), :]
y_t_b = bin_class(y_train[np.logical_or(y_train==1, y_train==5)], 1)
# x_train[np.logical_or(y_train==1, y_train==5), :].shape
x_test_ = x_test[np.logical_or(y_test==1, y_test==5), :]
y_test_b = bin_class(y_test[np.logical_or(y_test==1, y_test==5)], 1)

'''
Note we could also make this easier by setting select_process = np.logical_or(y_t
rain==1, y_train==5)
Thus, we then do x_train_ = x_train[select_process, :]
                 y_train_ = bin_class(y_train[select_process], 1)
'''

for C in C_list:
    clf = svm.SVC(C=C, kernel='poly', degree=Q, gamma=1, coef0=1)
    clf.fit(x_train_, y_t_b)

    y_t_b_ = clf.predict(x_train_)
    y_test_b_ = clf.predict(x_test_)

    E_in = np.sum(abs(y_t_b_ + y_t_b) == 0) / float(len(y_t_b_))
    E_out = np.sum(abs(y_test_b_ + y_test_b) == 0) / float(len(y_test_b_))
    print 'For C = %f, ' %C, 'Support vectors: ', len(clf.support_vectors_), 'E_i
n: ', E_in, 'E_out: ', E_out
```

```
For C = 0.001000,  Support vectors:  76 E_in:  0.00448430493274 E_out:  0.016509
4339623
For C = 0.010000,  Support vectors:  34 E_in:  0.00448430493274 E_out:  0.018867
9245283
For C = 0.100000,  Support vectors:  24 E_in:  0.00448430493274 E_out:  0.018867
9245283
For C = 1.000000,  Support vectors:  24 E_in:  0.00320307495195 E_out:  0.018867
9245283
```

Question 6

Answer Choice: **B**, When $C = 0.001$, the number of support vectors is lower at $Q = 5$
Reasoning: see code. We are doing a comparision between the two Q's so C has the outer loop. We are still studying a 1 versus 5 classifier. Note that the computational time slows down (increases) with each step forward in C_list; just an interesting thought, but I'm not entirely sure why.

We see E_{in} is larger for $Q = 2$ at $C = 0.0001$ and 0.01, ruling out choices *A* and *C*. We also see E_{out} is larger for $Q = 5$ at $C = 1$, ruling out *D*. At $C = 0.001$, there are more support vectors for $Q = 2$, meaning the number of support vectors is lower at $Q = 5$. So, our answer is *B*.

In [71]:
```
# parameters for kernel, SVM
C_list = [0.0001, 0.001, 0.01, 1]
Q_list = [2, 5]

print 'structure format: [Q=2, Q=5]'
for C in C_list:
    Q_E_in = []
    Q_E_out = []
    Q_sv = []
    for Q in Q_list:
        clf = svm.SVC(C=C, kernel='poly', degree=Q, gamma=1, coef0=1)
        clf.fit(x_train_, y_t_b)

        y_t_b_ = clf.predict(x_train_)
        y_test_b_ = clf.predict(x_test_)

        Q_E_in.append(np.sum(abs(y_t_b_ + y_t_b) == 0) / float(len(y_t_b_)))
        Q_E_out.append(np.sum(abs(y_test_b_ + y_test_b) == 0) / float(len(y_test_
b_)))
        Q_sv.append(len(clf.support_vectors_))

    print 'For C = %f' %C, 'E_in: ', [ '%.6f' % elem for elem in Q_E_in], \
          'E_out: ', [ '%.6f' % elem for elem in Q_E_out], 'Support vectors: ', Q
_sv
```

```
structure format: [Q=2, Q=5]
For C = 0.000100 E_in:  ['0.008969', '0.004484'] E_out:  ['0.016509', '0.018868'
] Support vectors:  [236, 26]
For C = 0.001000 E_in:  ['0.004484', '0.004484'] E_out:  ['0.016509', '0.021226'
] Support vectors:  [76, 25]
For C = 0.010000 E_in:  ['0.004484', '0.003844'] E_out:  ['0.018868', '0.021226'
] Support vectors:  [34, 23]
For C = 1.000000 E_in:  ['0.003203', '0.003203'] E_out:  ['0.018868', '0.021226'
] Support vectors:  [24, 21]
```

Question 7

Answer Choice: **B**, $C = 0.001$ is selected most often.
Reasoning: see code. I had a ton of trouble with importing as such: from sklearn.model_selection import cross_val_score from sklearn.model_selection import cross_val_predict

and I believe it was due to my scikit-learn version. I couldn't get the version to update properly, so I stuck with scikit-learn 0.18, and worked with the older version of sklearn using cross_validation, found from http://stackoverflow.com/questions/16379313/how-to-use-the-a-k-fold-cross-validation-in-scikit-with-naive-bayes-classifier-a (http://stackoverflow.com/questions/16379313/how-to-use-the-a-k-fold-cross-validation-in-scikit-with-naive-bayes-classifier-a) and https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/cross_validation.py (https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/cross_validation.py)

Now, originally I looped the function as 100 runs -> C_list -> 10 fold, but this did not seem optimized for time becuase the SVM had to be generated 400 times. Some of the time came from generating SVMs. The time ran for 1 run successfully was around 10 seconds, a for 10 runs was around 5 minutes. Thus, we rearranged the loop to C_list -> 100 runs -> 10 fold, which only needed to generate 4 SVMs. However, we could not avoid the fit process for all 10 cross folds..so this did take a while. Now, we saw it kept decreasing downwards in error and each run ran for ~40 minutes, which didn't seem to entirely match the prior questions in educational guessing and in speed, so I ran with the cross_validation built in score function, which outputs the score based on classification, and in our case with the SVM, the score was how many of the data points it got correct (as a fractional value). This was great, because we wanted to find how many we didn't get correct, so the Error was a simple 1 - score. And since the function is optimized in sklearn, the speed was ~20 seconds, which is orders of magnitude faster.

In [151]:
```
from sklearn.cross_validation import KFold, cross_val_score

# parameters for kernel, SVM, now with cross validation
C_list = [0.0001, 0.001, 0.01, 0.1, 1]
Q_list = 2
n_fold = 10

# had to re-run previous programs, so just re-ran the 1 versus 5 dataset extracti
on for soundness.
x_train_ = x_train[np.logical_or(y_train==1, y_train==5), :]
y_t_b = bin_class(y_train[np.logical_or(y_train==1, y_train==5)], 1)

C_E_in = []
for C in C_list:
    clf = svm.SVC(C=C, kernel='poly', degree=Q, gamma=1, coef0=1)

    C_i_E_in = []
    for r in xrange(100):
        # split into 10 fold cross validation, stores indices of [train], [test]
        k_fold = KFold(len(y_t_b), n_folds=10, shuffle=True)

        # take average of the 10 cv folds and compute E_in (function inherent in
cross_validation)
        # we note that the val_score varies based on classifier, for the SVM, it'
s a fraction of correct predictions
        # we just do 1 - frac(correct) to yield the desired error
        C_i_E_in.append(np.mean(1 - cross_val_score(clf, x_train_, y_t_b, cv=k_fo
ld, n_jobs=1)))

    C_E_in.append(C_i_E_in)
```

In [153]:
```
C_E_in = np.array(C_E_in).T
min_E_in = np.amin(C_E_in, axis=1)
min_C = np.argmin(C_E_in, axis=1)
```

In [154]:
```
import collections

# C_list values [0.0001, 0.001, 0.01, 0.1, 1] correspond to indices [0, 1, 2, 3, 4]
print collections.Counter(min_C)
```

```
Counter({1: 37, 2: 28, 3: 18, 4: 17})
```

Question 8

Answer Choice: **C**, 0.005
Reasoning: see code. The closest value in range is 0.005.

In [155]:
```
np.mean(C_E_in[np.where(min_C == 2)])
```

Out[155]:
```
0.0058023027927486398
```

Question 9

Answer Choice: **E**, $C = 10^6$
Reasoning: see code. Now, SVC uses rbf (radial basis function) kernel instead of the polynomial kernel. Again, we still use the same one-versus-one condition.

In [92]:
```
# parameters for kernel, SVM
C_list = [0.01, 1, 100, 10**4, 10**6]
Q_list = 2
n_fold = 10

for C in C_list:
    clf = svm.SVC(C=C, kernel='rbf', gamma=1)
    clf.fit(x_train_, y_t_b)

    y_t_b_ = clf.predict(x_train_)

    E_in = np.sum(abs(y_t_b_ + y_t_b) == 0) / float(len(y_t_b_))
    print 'E_in for C = %f: ' %C, E_in
```

```
E_in for C = 0.010000:  0.00384368994234
E_in for C = 1.000000:  0.00448430493274
E_in for C = 100.000000:  0.00320307495195
E_in for C = 10000.000000:  0.00256245996156
E_in for C = 1000000.000000:  0.000640614990391
```

Question 10

Answer Choice: **C**, 100
Reasoning: see code. It is interesting to note that for $C = 10^4, 10^6$ yields the same out of sample error. We run with higher C, and note that the error converges, which is what we expected!

In [94]:
```
# parameters for kernel, SVM
C_list = [0.01, 1, 100, 10**4, 10**6]
Q_list = 2
n_fold = 10

for C in C_list:
    clf = svm.SVC(C=C, kernel='rbf', gamma=1)
    clf.fit(x_train_, y_t_b)

    y_t_b_ = clf.predict(x_test_)

    E_out = np.sum(abs(y_t_b_ + y_test_b) == 0) / float(len(y_t_b_))
    print 'E_out for C = %f: ' %C, E_out
```

```
E_out for C = 0.010000:  0.0235849056604
E_out for C = 1.000000:  0.0212264150943
E_out for C = 100.000000:  0.0188679245283
E_out for C = 10000.000000:  0.0235849056604
E_out for C = 1000000.000000:  0.0235849056604
```

In [96]:
```
# parameters for kernel, SVM
C_list = [10**4, 10**6, 10**7, 10**8, 10**10, np.inf]
Q_list = 2
n_fold = 10

for C in C_list:
    clf = svm.SVC(C=C, kernel='rbf', gamma=1)
    clf.fit(x_train_, y_t_b)

    y_t_b_ = clf.predict(x_test_)

    E_out = np.sum(abs(y_t_b_ + y_test_b) == 0) / float(len(y_t_b_))
    print 'E_out for C = %f: ' %C, E_out
```

```
E_out for C = 10000.000000:  0.0235849056604
E_out for C = 1000000.000000:  0.0235849056604
E_out for C = 10000000.000000:  0.0259433962264
E_out for C = 100000000.000000:  0.0259433962264
E_out for C = 10000000000.000000:  0.0259433962264
E_out for C = inf:  0.0259433962264
```