In [92]:
```
import numpy as np
import random
import matplotlib.pyplot as plt
import sklearn.svm as svm
from sklearn.model_selection import KFold
```

In [52]:
```
#digit intensity symmetry
train = {}
for digit in range(10):
    train[digit] = []
with open('features.train','r') as f:
    for line in f:
        line = [float(i) for i in line.split()]
        train[line[0]].append(line[1:])
for digit in range(10):
    train[digit] = np.asarray(train[digit])
```

In [53]:
```
print(train[1].shape)
```

```
(1005, 2)
```

In [54]:
```
test = {}
for digit in range(10):
    test[digit] = []
with open('features.test','r') as f:
    for line in f:
        line = [float(i) for i in line.split()]
        test[line[0]].append(line[1:])
for digit in range(10):
    test[digit] = np.asarray(test[digit])
```

In [62]:
```
print([test[i].shape for i in range(10)])
```

```
[(359, 2), (264, 2), (198, 2), (166, 2), (200, 2), (160, 2), (170, 2),
(147, 2), (166, 2), (177, 2)]
```

In [55]:
```
def get_misclassified_point_idx(a,b):
    idx = []
    for i in range(len(a)):
        if a[i] != b[i]:
            idx.append(i)
    return idx
```

In [118]:
```
def soft_margin_svm(train_data,train_class,test_data,test_class,C,kernel
,degree=None):
    if kernel is 'poly':
        clf = svm.SVC(C = C, kernel = kernel, degree=degree, gamma=1,coe
f0 = 0)
    else:
        clf = svm.SVC(C = C, kernel = kernel,gamma=1)
    clf.fit(train_data, train_class)
    train_results = clf.predict(train_data)
    idx = get_misclassified_point_idx(train_class,train_results)
    e_in = len(idx)/float(len(train_class))
    test_results = clf.predict(test_data)
    idx = get_misclassified_point_idx(test_class,test_results)
    e_out = len(idx)/float(len(test_class))
    n_support_vecs = len(clf.support_)

    return e_in,e_out, n_support_vecs
```

In [119]:
```
def create_train_test_data_one_v_all(digit,train,test):
    train_data = []
    train_class = []
    test_data = []
    test_class = []

    for i in range(10):
        if i == digit:
            class_id = -1
        else:
            class_id = 1

        train_data.extend(train[i])
        train_class.extend([class_id for j in train[i]])
        test_data.extend(test[i])
        test_class.extend([class_id for j in test[i]])

    return np.asarray(train_data),np.asarray(train_class), np.asarray(te
st_data), np.asarray(test_class)
```

In [120]:
```
def create_train_test_data_one_v_one(digit_1,digit_2,train,test):
    train_data = []
    train_class = []
    test_data = []
    test_class = []

    for i in range(10):
        if i == digit_1:
            class_id = -1
        elif i == digit_2:
            class_id = 1
        else:
            continue

        train_data.extend(train[i])
        train_class.extend([class_id for j in train[i]])
        test_data.extend(test[i])
        test_class.extend([class_id for j in test[i]])

    return np.asarray(train_data),np.asarray(train_class), np.asarray(te
st_data), np.asarray(test_class)
```

In [121]:
```
#x vs all experiments for each digit x
C = 0.01
kernel = 'poly'
degree = 2
e_ins = []
e_outs = []
support_vecs = []
for digit in range(10):
    print('Digit: '+str(digit))
    train_data,train_class,test_data,test_class = create_train_test_data
_one_v_all(digit,train,test)
    e_in, e_out, n_support_vecs =  soft_margin_svm(train_data,train_clas
s,test_data,test_class,C,kernel,degree=degree)
    e_ins.append(e_in)
    e_outs.append(e_out)
    support_vecs.append(n_support_vecs)
    print('E_in: '+str(e_in))
    print('E_out: '+str(e_out))
    print('Support vectors: '+str(n_support_vecs))
```

```
Digit: 0
E_in: 0.119050884652
E_out: 0.128550074738
Support vectors: 2278
Digit: 1
E_in: 0.0142641612947
E_out: 0.0224215246637
Support vectors: 400
Digit: 2
E_in: 0.100260595254
E_out: 0.0986547085202
Support vectors: 1464
Digit: 3
E_in: 0.0902482512687
E_out: 0.0827105132038
Support vectors: 1320
Digit: 4
E_in: 0.0894253188863
E_out: 0.0996512207275
Support vectors: 1307
Digit: 5
E_in: 0.0762584007681
E_out: 0.079720976582
Support vectors: 1117
Digit: 6
E_in: 0.0910711836511
E_out: 0.0847035376183
Support vectors: 1330
Digit: 7
E_in: 0.0884652311068
E_out: 0.0732436472347
Support vectors: 1293
Digit: 8
E_in: 0.0743382252092
E_out: 0.0827105132038
Support vectors: 1091
Digit: 9
E_in: 0.0883280757098
E_out: 0.0881913303438
Support vectors: 1290
```

In [122]:
```
#1 vs 5 experiments
kernel = 'poly'
degree = 2
digit_1 = 1
digit_2 = 5
train_data,train_class,test_data,test_class = create_train_test_data_one
_v_one(digit_1,digit_2, train,test)
for C in [.0001,.001,.01,.1,1]:
    for degree in [2,5]:
        print('C: '+str(C)+', Q: '+str(degree))
        e_in, e_out, n_support_vecs =  soft_margin_svm(train_data,train_
class,test_data,test_class,C,kernel,degree=degree)
        print('E_in: '+str(e_in))
        print('E_out: '+str(e_out))
        print('Support vectors: '+str(n_support_vecs))
```

```
C: 0.0001, Q: 2
E_in: 0.0102498398463
E_out: 0.0165094339623
Support vectors: 244
C: 0.0001, Q: 5
E_in: 0.00448430493274
E_out: 0.0165094339623
Support vectors: 26
C: 0.001, Q: 2
E_in: 0.00448430493274
E_out: 0.0165094339623
Support vectors: 80
C: 0.001, Q: 5
E_in: 0.00448430493274
E_out: 0.0165094339623
Support vectors: 26
C: 0.01, Q: 2
E_in: 0.00448430493274
E_out: 0.0188679245283
Support vectors: 34
C: 0.01, Q: 5
E_in: 0.00512491992313
E_out: 0.0165094339623
Support vectors: 27
C: 0.1, Q: 2
E_in: 0.00448430493274
E_out: 0.0188679245283
Support vectors: 24
C: 0.1, Q: 5
E_in: 0.00448430493274
E_out: 0.0188679245283
Support vectors: 24
C: 1, Q: 2
E_in: 0.00384368994234
E_out: 0.0188679245283
Support vectors: 24
C: 1, Q: 5
E_in: 0.00448430493274
E_out: 0.0165094339623
Support vectors: 24
```

In [124]:
```
#Cross validation in 1 vs. 5
kernel = 'poly'
degree = 2
digit_1 = 1
digit_2 = 5
cv_train_data,cv_train_class,cv_test_data,cv_test_class = create_train_t
est_data_one_v_one(digit_1,digit_2, train,test)
kf = KFold(n_splits=10, shuffle=True)
kf.get_n_splits(train_data)
all_errors = []
selections = []
Cs = [0.0001, 0.001, 0.01, 0.1, 1]
for i in range(100):
    c_errors = []
    for C in Cs:
        done = False
        errors = []
        while len(errors) < 10:
            errors = []
            for train_index, test_index in kf.split(cv_train_data):
                train_data = cv_train_data[train_index]
                train_class = cv_train_class[train_index]
                if len(list(set(train_class))) < 2:
                    continue
                test_data = cv_train_data[test_index]
                test_class = cv_train_class[test_index]
                e_in, e_out, n_support_vecs =  soft_margin_svm(train_dat
a,train_class,test_data,test_class,C,kernel,degree=degree)
                errors.append(e_out)
        c_errors.append(np.mean(errors))
    selections.append(np.argmin(c_errors))
    all_errors.append(c_errors)
selected_idx = int(np.median(selections))
print('Selected C: '+str(Cs[selected_idx]))
print('Average error: '+str(np.mean([i[selected_idx] for i in all_errors
])))
print(list(set(selections)))
```

```
Selected C: 0.01
Average error: 0.00461273068757
[1, 2, 3, 4]
```

In [125]:
```
#1 vs 5 experiments
kernel = 'rbf'
train_data,train_class,test_data,test_class = create_train_test_data_one
_v_one(digit_1,digit_2, train,test)
for C in [.01,1,100,10**4, 10**6]:
    print('C: '+str(C))
    e_in, e_out, n_support_vecs =  soft_margin_svm(train_data,train_clas
s,test_data,test_class,C,kernel)
    print('E_in: '+str(e_in))
    print('E_out: '+str(e_out))
    print('Support vectors: '+str(n_support_vecs))
```

```
C: 0.01
E_in: 0.00384368994234
E_out: 0.0235849056604
Support vectors: 403
C: 1
E_in: 0.00448430493274
E_out: 0.0212264150943
Support vectors: 31
C: 100
E_in: 0.00320307495195
E_out: 0.0188679245283
Support vectors: 22
C: 10000
E_in: 0.00256245996156
E_out: 0.0235849056604
Support vectors: 20
C: 1000000
E_in: 0.000640614990391
E_out: 0.0235849056604
Support vectors: 17
```

In [ ]: