All the code can be accessed here and executed in google colab or any Jupyter Notebook environment: https://github.com/DragonBoy25830/caltech-cs-156

1) The hard margin-SVM problem can be solved through quadratic programming on all x_n where $n = \{1, 2, \cdots, N\}$. Each vector in the input space has d variables in it, but once we include our bias term, there are $d+1$ variables. As such, the original formulation of the hard-margin SVM problem is a quadratic programming w/ $d+1$ variables, so $\boxed{\text{the answer is d.}}$

2-6) The code below is used in all questions from 2-6.

```python
import numpy as np
from sklearn import svm
```
[1] ✓ 0.6s

```python
training_features = np.loadtxt("features.train")
testing_features = np.loadtxt("features.test")
```
[2] ✓ 0.0s

```python
def split_data(one_versus_all, dataset, choice1, choice2):
    to_return = dataset
    if not one_versus_all:
        to_return = []

        for digit, intensity, symmetry in dataset:
            if digit == choice1 or digit == choice2:
                to_return.append([digit, intensity, symmetry])

    to_return = np.array(to_return)
    digits = to_return[:, 0]
    intensity = to_return[:, 1]
    symmetry = to_return[:, 2]

    return digits, intensity, symmetry
```
[3] ✓ 0.0s

```python
def build_my_kernel(Q):
    def my_kernel(xn, xm):
        return (1 + np.dot(xn, xm.T)) ** Q

    return my_kernel
```
[4] ✓ 0.0s

```python
def generate_one_versus_all_yn(digits, one_choice):
    yn = np.ones(len(digits))

    for i, digit in enumerate(digits):
        if digit != one_choice:
            yn[i] = -1

    return np.array(yn)
```
[5] ✓ 0.0s

```python
def parse_digits(digits, choice1, choice2):
    result = []

    for digit in digits:
        if digit == choice1 or digit == choice2:
            result.append(digit)

    return digits
```
[6] ✓ 0.0s

```python
def generate_one_versus_one_yn(digits, choice1, choice2):
    parsed_digits = parse_digits(digits, choice1, choice2)
    yn = np.ones(len(parsed_digits))

    for i, digit in enumerate(digits):
        if digit == choice2:
            yn[i] = -1

    return yn
```
[7] ✓ 0.0s

```python
def calc_error(yn, y_pred):
    # misclassified points are opposite sign so will be -1 when multiplied
    error = yn * y_pred
    return np.count_nonzero(error == -1) / len(yn)
```
[8] ✓ 0.0s

```python
def run_svm(one_versus_all, choice, choice1=-1, C=0.01, Q=2):
    digits_train, intensities_train, symmetries_train = split_data(one_versus_all, training_features, choice, choice1)
    digits_test, intensities_test, symmetries_test = split_data(one_versus_all, testing_features, choice, choice1)

    if (one_versus_all):
        yn_train = generate_one_versus_all_yn(digits_train, choice)
        yn_test = generate_one_versus_all_yn(digits_test, choice)

    else:
        yn_train = generate_one_versus_one_yn(digits_train, choice, choice1)
        yn_test = generate_one_versus_one_yn(digits_test, choice, choice1)

    xn_train = np.column_stack((intensities_train, symmetries_train))
    xn_test= np.column_stack((intensities_test, symmetries_test))
    model = svm.SVC(C=C, kernel=build_my_kernel(Q), degree=Q, gamma=1)
    model.fit(xn_train, yn_train)

    y_train_pred = model.predict(xn_train)
    y_test_pred = model.predict(xn_test)

    Ein = calc_error(yn_train, np.array(y_train_pred))
    Eout = calc_error(yn_test, np.array(y_test_pred))

    return Ein, np.sum(model.n_support_), Eout
```
[9] ✓ 0.0s

2)

```python
choices = [0, 2, 4, 6, 8]

for choice in choices:
    Ein, _, _ = run_svm(True, choice)
    print(f"E_in for {choice} vs. all: {Ein}")
```
[11] ✓ 7.3s

```
E_in for 0 vs. all: 0.10588396653408312
E_in for 2 vs. all: 0.10026059525442327
E_in for 4 vs. all: 0.08942531888629818
E_in for 6 vs. all: 0.09107118365107666
E_in for 8 vs. all: 0.07433822520916199
```

The code above shows that the highest E_{in} is outputted by the 0 vs. all classifier, so the answer is a.

3)

Q3

```
choices = [1, 3, 5, 7, 9]

for choice in choices:
    Ein, _, _ = run_svm(True, choice)
    print(f"E_in for {choice} vs. all: {Ein}")
```

✓ 7.1s

```
E_in for 1 vs. all: 0.014401316691811822
E_in for 3 vs. all: 0.09024825126868742
E_in for 5 vs. all: 0.07625840076807022
E_in for 7 vs. all: 0.08846523110684405
E_in for 9 vs. all: 0.08832807570977919
```

The code above shows that the lowest E_{in} is outputted by the 1 vs. all classifier, so

the answer is a.

4)

Q4

```
choice_from_q2 = 0

_, n_svs, _ = run_svm(True, choice_from_q2)
print(f"# support vectors for {choice} vs. all: {n_svs}")
```

[13] ✓ 1.3s

```
# support vectors for 9 vs. all: 2179
```

```
choice_from_q3 = 1

_, n_svs, _ = run_svm(True, choice_from_q3)
print(f"# support vectors for {choice} vs. all: {n_svs}")
```

[14] ✓ 1.1s

```
# support vectors for 9 vs. all: 386
```

```
2179 - 386
```

[22] ✓ 0.0s

```
1793
```

The above code shows that the difference b/w the # of support vectors b/w the two classifiers is 1783 which is closest to 1800, so the answer is c.

5)

Q5

```
# investigate a and b
C = [0.001, 0.01, 0.1, 1]

for c in C:
    _, n_svs, _ = run_svm(False, 1, 5, c)
    print(f"# support vectors for C={c}: {n_svs}")
```

[15] ✓ 0.2s

```
# support vectors for C=0.001: 76
# support vectors for C=0.01: 34
# support vectors for C=0.1: 24
# support vectors for C=1: 24
```

```
# investigate c
for c in C:
    _, _, E_out = run_svm(False, 1, 5, c)
    print(f"E_out C={c}: {E_out}")
```

[16] ✓ 0.2s

```
E_out C=0.001: 0.01650943396226415
E_out C=0.01: 0.018867924528301886
E_out C=0.1: 0.018867924528301886
E_out C=1: 0.018867924528301886
```

```
# investigate d
for c in C:
    E_in, _, _ = run_svm(False, 1, 5, c)
    print(f"E_in C={c}: {E_in}")
```

[17] ✓ 0.2s

```
E_in C=0.001: 0.004484304932735426
E_in C=0.01: 0.004484304932735426
E_in C=0.1: 0.004484304932735426
E_in C=1: 0.0032030749519538757
```

From the above code, we see that a & b aren't true since the # of support vectors doesn't change from $C=0.1$ to $C=1$. We see that C is false since E_{out} increases as C increases. The code shows that d is true though since E_{in} is the smallest when $C=1$, so the answer is d.

6)

Q6

```
# investigate a
E_in, _, _ = run_svm(False, 1, 5, 0.0001)
print(f"E_in for C=0.0001 and Q=2: {E_in}")
E_in, _, _ = run_svm(False, 1, 5, 0.0001, 5)
```

6)

Q6

```python
# investigate a
E_in, _, _ = run_svm(False, 1, 5, 0.0001)
print(f"E_in for C=0.0001 and Q=2: {E_in}")
E_in, _, _ = run_svm(False, 1, 5, 0.0001, 5)
print(f"E_in for C=0.0001 and Q=5: {E_in}")
```

[18] ✓ 0.2s

```
E_in for C=0.0001 and Q=2: 0.008968609865470852
E_in for C=0.0001 and Q=5: 0.004484304932735426
```

```python
# investigate b
_, n_svm, _ = run_svm(False, 1, 5, 0.001)
print(f"# support vectors for C=0.001 and Q=2: {n_svm}")
_, n_svm, _ = run_svm(False, 1, 5, 0.001, 5)
print(f"# support vectors for C=0.001 and Q=5: {n_svm}")
```

[19] ✓ 0.1s

```
# support vectors for C=0.001 and Q=2: 76
# support vectors for C=0.001 and Q=5: 25
```

```python
# investigate c
E_in, _, _ = run_svm(False, 1, 5, 0.01)
print(f"E_in for C=0.01 and Q=2: {E_in}")
E_in, _, _ = run_svm(False, 1, 5, 0.01, 5)
print(f"E_in for C=0.01 and Q=5: {E_in}")
```

[20] ✓ 0.2s

```
E_in for C=0.01 and Q=2: 0.004484304932735426
E_in for C=0.01 and Q=5: 0.003843689942344651
```

```python
# investigate d
_, _, E_out = run_svm(False, 1, 5, 1)
print(f"E_out for C=1 and Q=2: {E_out}")
_, _, E_out = run_svm(False, 1, 5, 1, 5)
print(f"E_out for C=1 and Q=5: {E_out}")
```

[23] ✓ 0.7s

```
E_out for C=1 and Q=2: 0.018867924528301886
E_out for C=1 and Q=5: 0.02122641509433962
```

From the above code, we see that a, c, & d aren't true. We see that b is true since the # of support vectors is lower at Q=5 when C=0.001, so $\boxed{\text{the answer is b.}}$

7)

```python
import numpy as np
from sklearn import svm
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
```

[1] ✓ 0.6s

```python
training_features = np.loadtxt("features.train")
testing_features = np.loadtxt("features.test")
```

[2] ✓ 0.0s

```python
def split_data(one_versus_all, dataset, choice1, choice2):
    to_return = dataset
    if not one_versus_all:
        to_return = []

        for digit, intensity, symmetry in dataset:
            if digit == choice1 or digit == choice2:
                to_return.append([digit, intensity, symmetry])

    to_return = np.array(to_return)
    digits = to_return[:, 0]
    intensity = to_return[:, 1]
    symmetry = to_return[:, 2]

    return digits, intensity, symmetry
```

[3] ✓ 0.0s

```python
def build_my_kernel(Q):
    def my_kernel(xn, xm):
        return (1 + np.dot(xn, xm.T)) ** Q

    return my_kernel
```
[4] ✓ 0.0s

```python
def generate_one_versus_all_yn(digits, one_choice):
    yn = np.ones(len(digits))

    for i, digit in enumerate(digits):
        if digit != one_choice:
            yn[i] = -1

    return np.array(yn)
```
[5] ✓ 0.0s

```python
def parse_digits(digits, choice1, choice2):
    result = []

    for digit in digits:
        if digit == choice1 or digit == choice2:
            result.append(digit)

    return digits
```
[6] ✓ 0.0s

```python
def generate_one_versus_one_yn(digits, choice1, choice2):
    parsed_digits = parse_digits(digits, choice1, choice2)
    yn = np.ones(len(parsed_digits))

    for i, digit in enumerate(digits):
        if digit == choice2:
            yn[i] = -1

    return yn
```
[7] ✓ 0.0s

```python
def calc_error(yn, y_pred):
    # misclassified points are opposite sign so will be -1 when multiplied
    error = yn * y_pred
    return np.count_nonzero(error == -1) / len(yn)
```
[8] ✓ 0.0s

```python
def run_svm(one_versus_all, choice, choice1=-1, C=0.01, Q=2):
    digits_train, intensities_train, symmetries_train = split_data(one_versus_all, training_features, choice, choice1)
    digits_test, intensities_test, symmetries_test = split_data(one_versus_all, testing_features, choice, choice1)

    if (one_versus_all):
        yn_train = generate_one_versus_all_yn(digits_train, choice)
        yn_test = generate_one_versus_all_yn(digits_test, choice)

    else:
        yn_train = generate_one_versus_one_yn(digits_train, choice, choice1)
        yn_test = generate_one_versus_one_yn(digits_test, choice, choice1)

    xn_train = np.column_stack((intensities_train, symmetries_train))
    xn_test= np.column_stack((intensities_test, symmetries_test))
    model = svm.SVC(C=C, kernel=build_my_kernel(Q), degree=Q, gamma=1)
    model.fit(xn_train, yn_train)

    y_train_pred = model.predict(xn_train)
    y_test_pred = model.predict(xn_test)

    error = cross_val_score(model, xn_train, yn_train)

    return error
```
[27] ✓ 0.0s

```
C = [0.0001, 0.001, 0.01, 0.1, 1]
result = [0, 0, 0, 0, 0]

for i in range(100):
    Ecv_arr = []

    for c in C:
        Ecv = run_svm(False, 1, 5, c)
        Ecv_arr.append(Ecv)

    result[np.argmin(Ecv_arr)] += 1

result
```
[28] ✓ 1m 5.3s

```
[0, 0, 0, 100, 0]
```

The code above shows that C=0.1 is chosen the most, so the answer is d.

8)

```
Ecv_arr = []

for i in range(100):
    for c in C:
        Ecv = run_svm(False, 1, 5, 0.01)
        Ecv_arr.append(Ecv)

np.mean(Ecv_arr)
```
[29] ✓ 1m 4.9s

```
0.9955148685180635
```

```
1 - 0.9955148685180635
```
[30] ✓ 0.0s

```
0.004485131481936522
```

The average value is closest to 0.005, so the answer is c.

9-10) The below code will be used for question 9 & 10:

```python
import numpy as np
from sklearn import svm
```

[62] ✓ 0.0s

```python
import numpy as np
from sklearn import svm
```

[63] ✓ 0.0s

```python
training_features = np.loadtxt("features.train")
testing_features = np.loadtxt("features.test")
```

[64] ✓ 0.0s

```python
def build_my_kernel(Q):
    def my_kernel(xn, xm):
        return (1 + np.dot(xn, xm.T)) ** Q

    return my_kernel
```

[65] ✓ 0.0s

```python
def generate_one_versus_all_yn(digits, one_choice):
    yn = np.ones(len(digits))

    for i, digit in enumerate(digits):
        if digit != one_choice:
            yn[i] = -1

    return np.array(yn)
```

[66] ✓ 0.0s

```python
def parse_digits(digits, choice1, choice2):
    result = []

    for digit in digits:
        if digit == choice1 or digit == choice2:
            result.append(digit)

    return digits
```

[67] ✓ 0.0s

```python
def generate_one_versus_one_yn(digits, choice1, choice2):
    parsed_digits = parse_digits(digits, choice1, choice2)
    yn = np.ones(len(parsed_digits))

    for i, digit in enumerate(digits):
        if digit == choice2:
            yn[i] = -1

    return yn
```

[68] ✓ 0.0s

```python
def calc_error(yn, y_pred):
    # misclassified points are opposite sign so will be -1 when multiplied
    error = yn * y_pred
    return np.count_nonzero(error == -1) / len(yn)
```

[69] ✓ 0.0s

```python
def split_data(one_versus_all, dataset, choice1, choice2):
    to_return = dataset
    if not one_versus_all:
        to_return = []

        for digit, intensity, symmetry in dataset:
            if digit == choice1 or digit == choice2:
                to_return.append([digit, intensity, symmetry])

    to_return = np.array(to_return)
    digits = to_return[:, 0]
    intensity = to_return[:, 1]
    symmetry = to_return[:, 2]

    return digits, intensity, symmetry
```
[70] ✓ 0.0s

```python
def my_kernel(xn, xm):
    return np.exp(-1 * np.linalg.norm(xn - xm) ** 2)
```
[71] ✓ 0.0s

```python
def generate_one_versus_all_yn(digits, one_choice):
    yn = np.ones(len(digits))

    for i, digit in enumerate(digits):
        if digit != one_choice:
            yn[i] = -1

    return np.array(yn)
```
[73] ✓ 0.0s

```python
def parse_digits(digits, choice1, choice2):
    result = []

    for digit in digits:
        if digit == choice1 or digit == choice2:
            result.append(digit)

    return digits
```
[74] ✓ 0.0s

```python
def generate_one_versus_one_yn(digits, choice1, choice2):
    parsed_digits = parse_digits(digits, choice1, choice2)
    yn = np.ones(len(parsed_digits))

    for i, digit in enumerate(digits):
        if digit == choice2:
            yn[i] = -1

    return yn
```
[75] ✓ 0.0s

```python
def calc_error(yn, y_pred):
    # misclassified points are opposite sign so will be -1 when multiplied
    error = yn * y_pred
    return np.count_nonzero(error == -1) / len(yn)
```
[76] ✓ 0.0s

```python
def run_svm(one_versus_all, choice, choice1=-1, C=0.01, Q=2):
    digits_train, intensities_train, symmetries_train = split_data(one_versus_all, training_features, choice, choice1)
    digits_test, intensities_test, symmetries_test = split_data(one_versus_all, testing_features, choice, choice1)

    if (one_versus_all):
        yn_train = generate_one_versus_all_yn(digits_train, choice)
        yn_test = generate_one_versus_all_yn(digits_test, choice)

    else:
        yn_train = generate_one_versus_one_yn(digits_train, choice, choice1)
        yn_test = generate_one_versus_one_yn(digits_test, choice, choice1)

    xn_train = np.column_stack((intensities_train, symmetries_train))
    xn_test= np.column_stack((intensities_test, symmetries_test))
    model = svm.SVC(C=C, kernel='rbf', gamma=1)
    model.fit(xn_train, yn_train)

    y_train_pred = model.predict(xn_train)
    y_test_pred = model.predict(xn_test)

    Ein = calc_error(yn_train, np.array(y_train_pred))
    Eout = calc_error(yn_test, np.array(y_test_pred))

    return Ein, np.sum(model.n_support_), Eout
```
[81] ✓ 0.0s

9)

Q9

```
C = [0.01, 1, 100, 1e4, 1e6]

for c in C:
    E_in, _, _ = run_svm(False, 1, 5, c)
    print(f"E_in for C={c}: {E_in}")
```

[82] ✓ 0.2s

```
E_in for C=0.01: 0.003843689942344651
E_in for C=1: 0.004484304932735426
E_in for C=100: 0.0032030749519538757
E_in for C=10000.0: 0.0025624599615631004
E_in for C=1000000.0: 0.0006406149903907751
```

The above code shows that a C value of 10^6 minimizes E_{in}, so the answer is e

10)

Q10

```
for c in C:
    _, _, E_out = run_svm(False, 1, 5, c)
    print(f"E_out for C={c}: {E_out}")
```

[83] ✓ 0.2s

```
E_out for C=0.01: 0.02358490566037736
E_out for C=1: 0.02122641509433962
E_out for C=100: 0.018867924528301886
E_out for C=10000.0: 0.02358490566037736
E_out for C=1000000.0: 0.02358490566037736
```

The above code shows that a C value of 100 minimizes E_{out} so the answer is c