1. hard-margin SVM problem (minimize $\frac{1}{2} w^T w$)

subject to $y_n(w^T x_n + b) \geq 1$ (lec. 14, slide 11)

x_n and y_n are inputs, so w^T and b are the variables involved

w^T has d weights, but b is an additional variable (w_0) $\rightarrow$ $d+1$ variables

[a] a quadratic programming problem w/ $d+1$ variables

2.


```python
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

#each row is: digit, intensity, symmetry
train = np.loadtxt('features.train')
x_train = train[:,1:]
digits_train = train[:,0]
test = np.loadtxt('features.test')
x_test = test[:,1:]
digits_test = test[:,0]

compare = [0, 2, 4, 6, 8]
highest_ein = 0
worst_model = None
for digit in compare:
    y_train = np.where(digits_train == digit, 1, -1)
    y_test = np.where(digits_test == digit, 1, -1)
    model = SVC(C=0.01, kernel='poly', degree=2, gamma=1.0, coef0=1.0)
    model.fit(x_train, y_train)
    #y_pred = model.predict(x_train)
    #print(y_pred)
    #error = np.sum(y_pred != y_train) / len(y_train)
    error = 1 - model.score(x_train, y_train)
    if error > highest_ein:
        highest_ein = error
        worst_model = model
    print('error for ', digit, ' vs all: ', error, '\n')
```

```
error for  0  vs all:  0.10588396653408316

error for  2  vs all:  0.10026059525442321

error for  4  vs all:  0.08942531888629812

error for  6  vs all:  0.09107118365107669

error for  8  vs all:  0.074338225209162
```

For $C=0.01$ and $Q=2$, the 0 vs. all classifier has the highest E_{in} of $0.106 \Rightarrow$ [a]

3.

```
train = np.loadtxt('features.train')
x_train = train[:,1:]
digits_train = train[:,0]
test = np.loadtxt('features.test')
x_test = test[:,1:]
digits_test = test[:,0]

compare = [1, 3, 5, 7, 9]
lowest_ein = 1
best_model = None
for digit in compare:
    y_train = np.where(digits_train == digit, 1, -1)
    y_test = np.where(digits_test == digit, 1, -1)
    model = SVC(C=0.01, kernel='poly', degree=2, gamma=1.0, coef0=1.0)
    model.fit(x_train, y_train)
    #y_pred = model.predict(x_train)
    #print(y_pred)
    #error = np.sum(y_pred != y_train) / len(y_train)
    error = 1 - model.score(x_train, y_train)
    if error < lowest_ein:
        lowest_ein = error
        best_model = model
    print('error for ', digit, ' vs all: ', error, '\n')
```

```
error for  1  vs all:  0.014401316691811772

error for  3  vs all:  0.09024825126868741

error for  5  vs all:  0.07625840076807022

error for  7  vs all:  0.08846523110684401

error for  9  vs all:  0.08832807570977919
```

For $C=0.01$ and $Q=2$, the lowest E_{in} of 0.014 is the 1 vs. all classifier => [a]

4.

```
[ ] difference = abs(sum(list(best_model.n_support_)) - sum(list(worst_model.n_support_)))
    print('difference in number of support vectors: ', difference)
```

```
difference in number of support vectors:  1793
```

For the model with highest E_{in} in (2) and lowest E_{in} in (3), the difference in number of support vectors is 1793 => [c] 1800

5.

```
train = np.loadtxt('features.train')
x_train = train[:,1:]
digits_train = train[:,0]
test = np.loadtxt('features.test')
x_test = test[:,1:]
digits_test = test[:,0]

C_vals = [0.001, 0.01, 0.1, 1]
train_filter = (train[:, 0] == 1) | (train[:, 0] == 5)
x_train = train[train_filter, 1:]

y_train = np.where(train[train_filter, 0] == 1, 1, -1)


test_filter = (test[:, 0] == 1) | (test[:, 0] == 5)
x_test = test[test_filter, 1:]
y_test = np.where(test[test_filter, 0] == 1, 1, -1)


highest_ein = 0
worst_model = None
for val in C_vals:
    model = SVC(C=val, kernel='poly', degree=2, gamma=1.0, coef0=1.0)
    model.fit(x_train, y_train)
    y_pred = model.predict(x_train)
    in_error = 1 - model.score(x_train, y_train)
    y_pred_out = model.predict(x_test)
    out_error = 1 - model.score(x_test, y_test)
    num = sum(list(model.n_support_))
    print("for C =", val, "E_in =", in_error, "E_out =", out_error, "num support vectors =", num, "\n")
```

```
for C = 0.001 E_in = 0.004484304932735439 E_out = 0.01650943396226412 num support vectors = 76

for C = 0.01 E_in = 0.004484304932735439 E_out = 0.018867924528301883 num support vectors = 34

for C = 0.1 E_in = 0.004484304932735439 E_out = 0.018867924528301883 num support vectors = 24

for C = 1 E_in = 0.0032030749519538215 E_out = 0.018867924528301883 num support vectors = 24
```

Based on my code output, of the answer choices, the maximum C achieves the lowest E_{in} is the only true one => [d]

6.

```
train = np.loadtxt('features.train')
x_train = train[:,1:]
digits_train = train[:,0]
test = np.loadtxt('features.test')
x_test = test[:,1:]
digits_test = test[:,0]

C_vals = [0.0001, 0.001, 0.01, 0.1, 1]
Q_vals = [2, 5]
train_filter = (train[:, 0] == 1) | (train[:, 0] == 5)
x_train = train[train_filter, 1:]
y_train = np.where(train[train_filter, 0] == 1, 1, -1)


test_filter = (test[:, 0] == 1) | (test[:, 0] == 5)
x_test = test[test_filter, 1:]
y_test = np.where(test[test_filter, 0] == 1, 1, -1)

for val in C_vals:
  for q in Q_vals:
    model = SVC(C=val, kernel='poly', degree=q, gamma=1.0, coef0=1.0)
    model.fit(x_train, y_train)
    y_pred = model.predict(x_train)
    in_error = np.sum(y_pred != y_train) / len(y_train)
    y_pred_out = model.predict(x_test)
    out_error = np.sum(y_pred_out != y_test) / len(y_test)
    num = len(model.support_vectors_)
    print("for C =", val, "and Q =", q, "E_in =", in_error, "E_out =", out_error, "num support vectors =", num, "\n")
```

```
for C = 0.0001 and Q = 2 E_in = 0.008968609865470852 E_out = 0.01650943396226415 num support vectors = 236

for C = 0.0001 and Q = 5 E_in = 0.004484304932735426 E_out = 0.018867924528301886 num support vectors = 26

for C = 0.001 and Q = 2 E_in = 0.004484304932735426 E_out = 0.01650943396226415 num support vectors = 76

for C = 0.001 and Q = 5 E_in = 0.004484304932735426 E_out = 0.02122641509433962 num support vectors = 25

for C = 0.01 and Q = 2 E_in = 0.004484304932735426 E_out = 0.018867924528301886 num support vectors = 34

for C = 0.01 and Q = 5 E_in = 0.003843689942344651 E_out = 0.02122641509433962 num support vectors = 23

for C = 0.1 and Q = 2 E_in = 0.004484304932735426 E_out = 0.018867924528301886 num support vectors = 24

for C = 0.1 and Q = 5 E_in = 0.0032030749519538757 E_out = 0.018867924528301886 num support vectors = 25

for C = 1 and Q = 2 E_in = 0.0032030749519538757 E_out = 0.018867924528301886 num support vectors = 24

for C = 1 and Q = 5 E_in = 0.0032030749519538757 E_out = 0.02122641509433962 num support vectors = 21
```

Based on my code output, of the answer choices, when $C=0.001$, the number of support vectors is lower at $Q=5$ is the only true one => [b]

7.

```
C_vals = [0.0001, 0.001, 0.01, 0.1, 1]
chosen = defaultdict(int)
avg_ev = defaultdict(float)
train_filter = (train[:, 0] == 1) | (train[:, 0] == 5)
x_train = train[train_filter, 1:]
y_train = np.where(train[train_filter, 0] == 1, 1, -1)
test_filter = (test[:, 0] == 1) | (test[:, 0] == 5)
x_test = test[test_filter, 1:]
y_test = np.where(test[test_filter, 0] == 1, 1, -1)
minE = 1
bestC = None
for _ in range(100):
  minE = 1
  bestC = None
  data = list(zip(x_train, y_train))
  random.shuffle(data)
  x_train, y_train = zip(*data)
  for i in range(len(C_vals)):
    model = SVC(C=C_vals[i], kernel='poly', degree=2, gamma=1.0, coef0=1.0)
    cv_scores = cross_val_score(model, x_train, y_train, cv=10)
    E_ev = 1 - np.mean(cv_scores)
    avg_ev[C_vals[i]] += E_ev
    if E_ev < minE:
      minE = E_ev
      bestC = C_vals[i]
    elif E_ev == minE:
      bestC = min(bestC, C_vals[i])
  chosen[bestC] += 1
max_c = max(chosen, key=chosen.get)
print(max_c)
print(avg_ev[max_c]/100)
```

```
0.001
0.004733831455169013
```

Based on my code, $C = 0.001$ is selected most often => [b]

8. Based on my code above from (7), the average value of E_{cv} is about 0.0047, closest to 0.005 => [c]

9.

```
C_vals = [0.01, 1, 100, 10**4, 10**6]
train_filter = (train[:, 0] == 1) | (train[:, 0] == 5)
x_train = train[train_filter, 1:]
y_train = np.where(train[train_filter, 0] == 1, 1, -1)
test_filter = (test[:, 0] == 1) | (test[:, 0] == 5)
x_test = test[test_filter, 1:]
y_test = np.where(test[test_filter, 0] == 1, 1, -1)
minE_in = 1
bestC_in = None
minE_out = 1
bestC_out = None
for i in range(len(C_vals)):
  model = SVC(C=C_vals[i], kernel='rbf', gamma=1.0, coef0=1.0)
  model.fit(x_train, y_train)
  y_pred = model.predict(x_train)
  in_error = np.sum(y_pred != y_train) / len(y_train)
  y_pred_out = model.predict(x_test)
  out_error = np.sum(y_pred_out != y_test) / len(y_test)
  if in_error < minE_in:
    minE_in = in_error
    bestC_in = C_vals[i]
  if out_error < minE_out:
    minE_out = out_error
    bestC_out = C_vals[i]
print("C for lowest E_in: ", bestC_in)
print("C for lowest E_out: ", bestC_out)
```

```
C for lowest E_in:  1000000
C for lowest E_out:  100
```

Based on my code, the C that results in the lowest E_{in} is 10^6 => [e]

10. Based on my code above from (9), the C that results in the lowest E_{out} is 100 => [c]