Assignment 1 Part III

November 13, 2023

[ ]:
```
import numpy as np
import scipy.optimize as sco
import matplotlib.pyplot as plt
```

0.1 PART III

Data generation

[ ]:
```
# TRAIN

train_samples_per_class = 30

# Set the variance for each class
variance = 0.3

n1a = np.random.normal(loc=[0, 0], scale=np.sqrt(variance),␣
 ↪size=(train_samples_per_class, 2))
n1b = np.random.normal(loc=[0, 1], scale=np.sqrt(variance),␣
 ↪size=(train_samples_per_class, 2))
n2a = np.random.normal(loc=[1, 1], scale=np.sqrt(variance),␣
 ↪size=(train_samples_per_class, 2))
n2b = np.random.normal(loc=[1, 0], scale=np.sqrt(variance),␣
 ↪size=(train_samples_per_class, 2))

labels = np.array([1]*train_samples_per_class*2 +␣
 ↪[-1]*train_samples_per_class*2)


X_train = np.vstack([n1a, n1b, n2a, n2b])
Y_train = labels

shuffle_idx = np.random.permutation(X_train.shape[0])
X_train = X_train[shuffle_idx]
Y_train = Y_train[shuffle_idx]


# TEST
```

```
test_samples_per_class = 200

n1a = np.random.normal(loc=[0, 0], scale=np.sqrt(variance),␣
 ↪size=(test_samples_per_class, 2))
n1b = np.random.normal(loc=[0, 1], scale=np.sqrt(variance),␣
 ↪size=(test_samples_per_class, 2))
n2a = np.random.normal(loc=[1, 1], scale=np.sqrt(variance),␣
 ↪size=(test_samples_per_class, 2))
n2b = np.random.normal(loc=[1, 0], scale=np.sqrt(variance),␣
 ↪size=(test_samples_per_class, 2))

labels = np.array([1]*test_samples_per_class*2 + [-1]*test_samples_per_class*2)


X_test = np.vstack([n1a, n1b, n2a, n2b])
Y_test = labels

shuffle_idx = np.random.permutation(X_test.shape[0])
X_test = X_test[shuffle_idx]
Y_test = Y_test[shuffle_idx]
```

[ ]:
```
def soft_margin_SVM_fit(X, Y, C=1.0):
    #zeta >= 1 - y_i(w^T x_i + b)
    #MINIMIZE: 0.5*||w||^2 + C*sum(zeta_i)

    d = X.shape[1] # d dimensions


    kernel = lambda params: 0.5*np.linalg.norm(params[:-1])**2 + C * np.sum(np.
 ↪maximum(0, 1 - Y * (np.matmul(X, params[:-1]) - params[-1])))

    return sco.minimize(kernel, np.zeros(d+1))
```

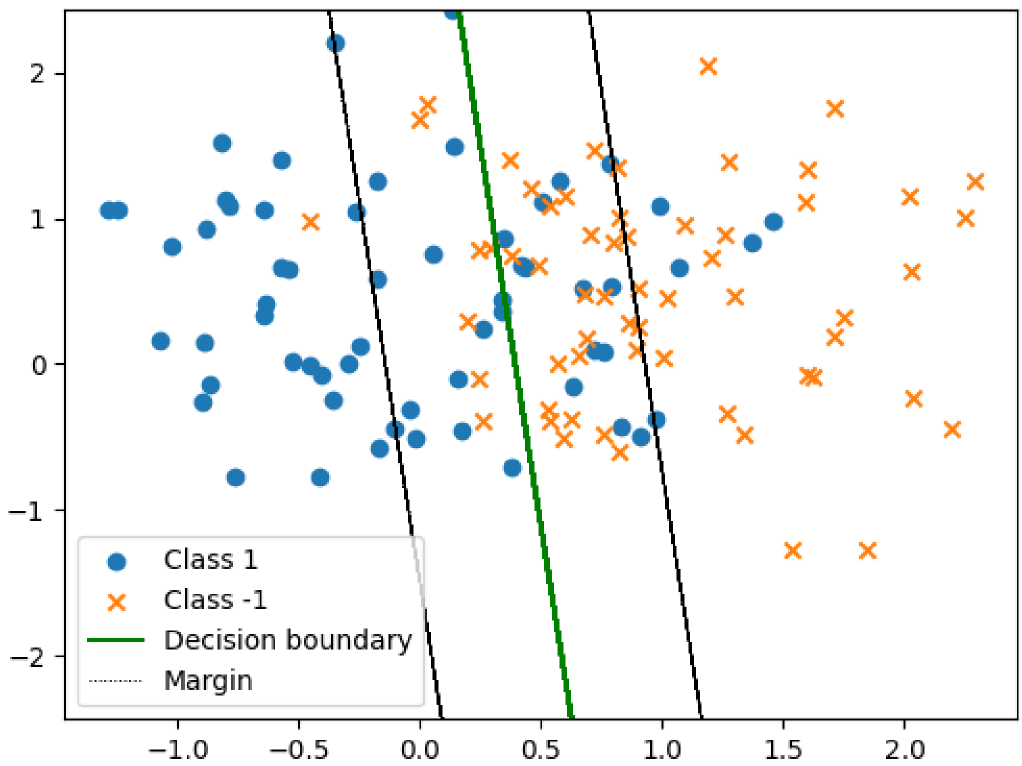
[ ]:
```
params = soft_margin_SVM_fit(X_train, Y_train, C = 10)
params = params.x
```

[ ]:
```
params
```

[ ]:
```
array([-1.86628952, -0.17877441, -0.73678749])
```

[ ]:
```
# plt.plot(X_train[:, 0], X_train[:, 1], 'o')
plt.scatter(X_train[Y_train == 1][:, 0], X_train[Y_train == 1][:, 1],␣
 ↪label='Class 1', marker='o')
```

```
plt.scatter(X_train[Y_train == -1][:, 0], X_train[Y_train == -1][:, 1],␣
    ↪label='Class -1', marker='x')
plt.plot(X_train[:, 0], -params[0]*X_train[:, 0]/params[1] + params[-1]/
    ↪params[1], label='Decision boundary', color='green')
plt.plot(X_train[:, 0], -params[0]*X_train[:, 0]/params[1] + params[-1]/
    ↪params[1] + 1/params[1], color='black', ls=':', lw=0.7)
plt.plot(X_train[:, 0], -params[0]*X_train[:, 0]/params[1] + params[-1]/
    ↪params[1] - 1/params[1], color='black', ls=':', lw=0.7, label='Margin')
plt.ylim(-max(abs(X_train[:, 1])), max(abs(X_train[:, 1])))
plt.legend()
plt.show()
```



[ ]:
```
Y_pred = np.sign(np.dot(X_test, params[:-1]) - params[-1])
```

[ ]:
```
accuracy = np.count_nonzero(Y_pred == Y_test) / Y_test.shape[0]

print('Accuracy: ', accuracy*100)
```

```
Accuracy:  80.375
```

[ ]:
```
hold_out_rho = 0.3
C = 5
```

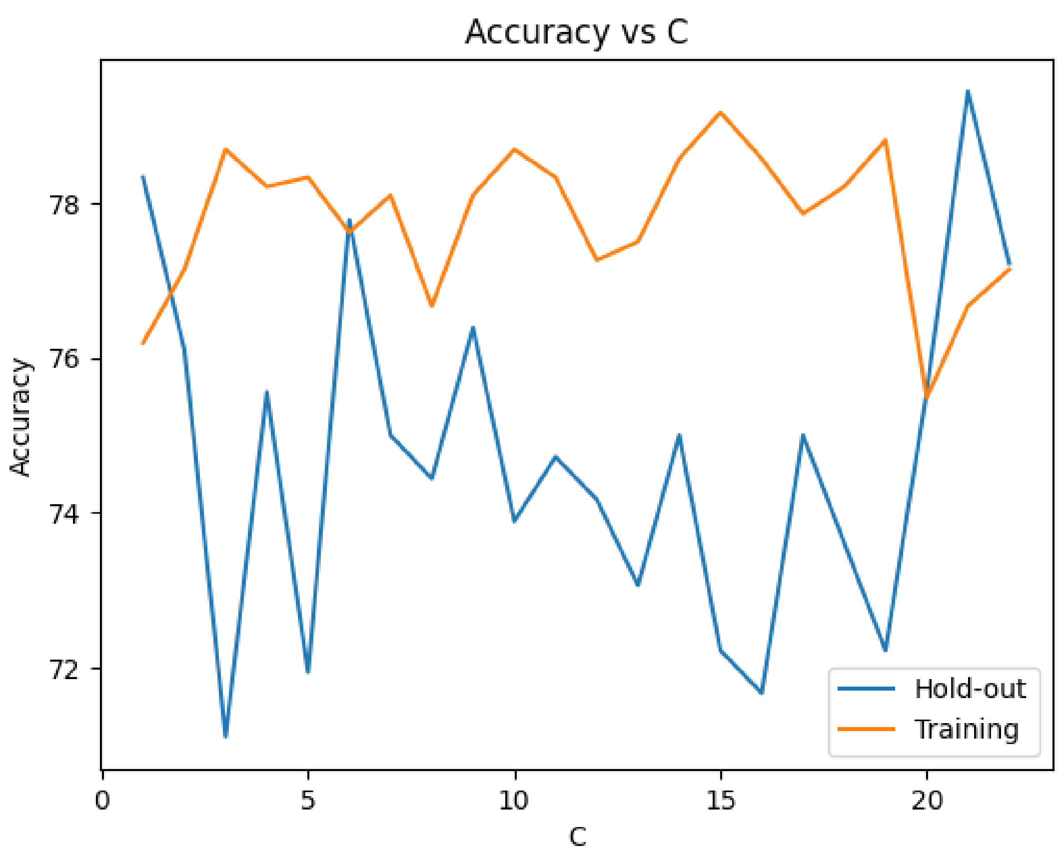
```
n_rep = 10
def outCVSVM(X, Y, C, n_rep, hold_out_rho):
    n = len(X)
    n_hold_out = int(n*hold_out_rho)
    hold_out_accuracy = np.zeros(n_rep)
    train_accuracy = np.zeros(n_rep)
    for i in range(n_rep):
        shuffle_idx = np.random.permutation(n)
        X_train = X[shuffle_idx]
        Y_train = Y[shuffle_idx]
        X_hold_out = X_train[:n_hold_out]
        Y_hold_out = Y_train[:n_hold_out]
        X_train = X_train[n_hold_out:]
        Y_train = Y_train[n_hold_out:]
        # Y_pred_hold_out = soft_margin_SVM_fit(X_train, Y_train, X_hold_out, k)
        params = soft_margin_SVM_fit(X_train, Y_train, C).x
        Y_pred_hold_out = np.sign(np.dot(X_hold_out, params[:-1]) - params[-1])
        hold_out_accuracy[i] = np.count_nonzero(Y_hold_out == Y_pred_hold_out)/
 ↪len(Y_hold_out)
        # Y_pred_train = soft_margin_SVM_fit(X_train, Y_train, X_train, k)
        Y_pred_train = np.sign(np.dot(X_train, params[:-1]) - params[-1])
        train_accuracy[i] = np.count_nonzero(Y_train == Y_pred_train)/
 ↪len(Y_train)


    return np.round([np.mean(hold_out_accuracy)*100, np.
 ↪mean(train_accuracy)*100], 2)
```

[ ]:
```
aaa = []
for C in range(1, 23):
    svmout = outCVSVM(X_train, Y_train, C, n_rep, hold_out_rho)
    print("C = ", C, ": ", svmout)
    aaa.append(svmout)
```

```
C =  1 :  [78.33 76.19]
C =  2 :  [76.11 77.14]
C =  3 :  [71.11 78.69]
C =  4 :  [75.56 78.21]
C =  5 :  [71.94 78.33]
C =  6 :  [77.78 77.62]
C =  7 :  [75.   78.1]
C =  8 :  [74.44 76.67]
C =  9 :  [76.39 78.1 ]
C =  10 :  [73.89 78.69]
C =  11 :  [74.72 78.33]
C =  12 :  [74.17 77.26]
C =  13 :  [73.06 77.5 ]
```

```
C =  14 :  [75.   78.57]
C =  15 :  [72.22 79.17]
C =  16 :  [71.67 78.57]
C =  17 :  [75.   77.86]
C =  18 :  [73.61 78.21]
C =  19 :  [72.22 78.81]
C =  20 :  [75.56 75.48]
C =  21 :  [79.44 76.67]
C =  22 :  [77.22 77.14]
```

[ ]:
```python
plt.plot(range(1, 23), [x[0] for x in aaa], label="Hold-out")
plt.plot(range(1, 23), [x[1] for x in aaa], label="Training")
plt.xlabel("C")
plt.ylabel("Accuracy")
plt.title("Accuracy vs C")
plt.legend()
plt.show()
```



[ ]:
```python
C_best = np.argmax([x[0] for x in aaa])
print('C =', C_best, 'is the best C for hold-out')
```

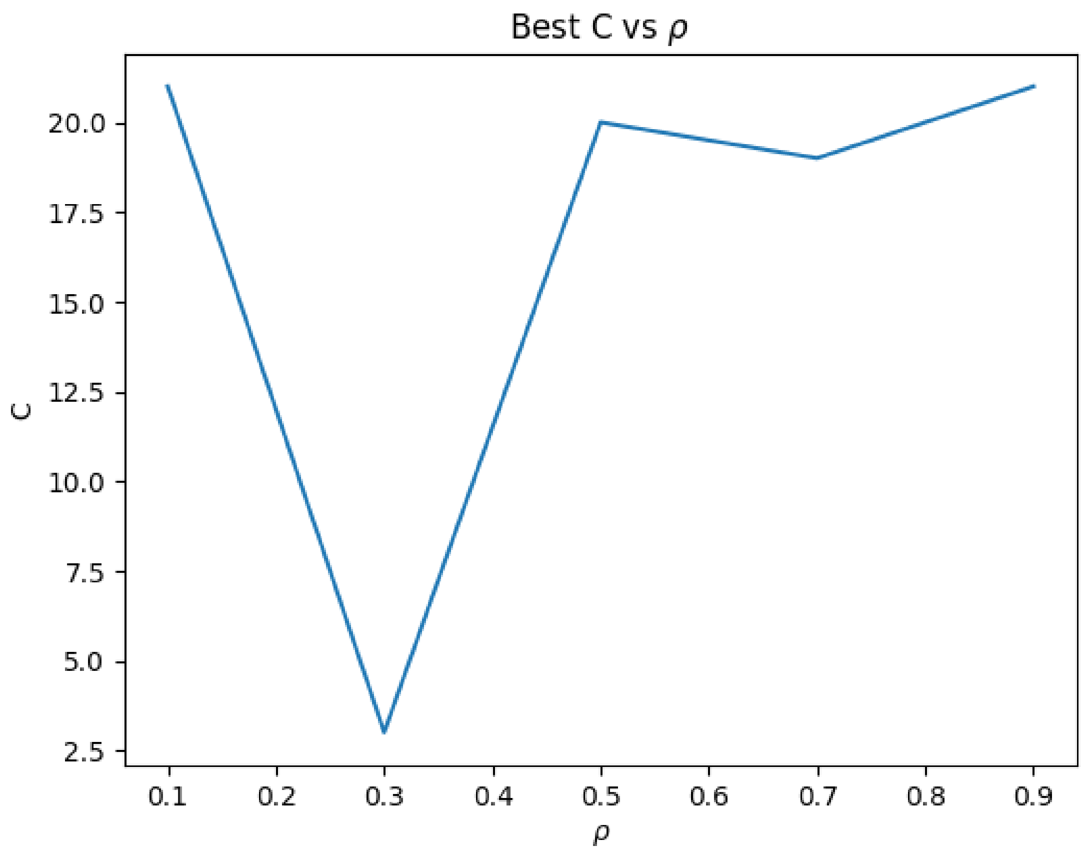
```
C = 20 is the best C for hold-out
```

(b) How is the value of C affected by (percentage of points held out) and number of repetitions? What does a large number of repetitions provide?

```
[ ]: # Rho vs C
best_C = []
for rho in [0.1, 0.3, 0.5, 0.7, 0.9]:
    # print("rho = ", rho, ": ", outCVSVM(X_train, Y_train, 5, 10, rho))
    aaa = []
    for C in range(1, 23):
        svmout = outCVSVM(X_train, Y_train, C, n_rep, hold_out_rho)
        aaa.append(svmout)
    best_C.append(np.argmax([x[0] for x in aaa]))
    print('Rho = ' + str(rho) + ': C =', np.argmax([x[0] for x in aaa]), 'is␣
↪the best C for hold-out')
```

```
Rho = 0.1: C = 21 is the best C for hold-out
Rho = 0.3: C = 3 is the best C for hold-out
Rho = 0.5: C = 20 is the best C for hold-out
Rho = 0.7: C = 19 is the best C for hold-out
Rho = 0.9: C = 21 is the best C for hold-out
```

```
[ ]: plt.plot([0.1, 0.3, 0.5, 0.7, 0.9], best_C)
plt.xlabel("$\\rho$")
plt.ylabel("C")
plt.title("Best C vs $\\rho$")
plt.show()
```



[ ]:
```
for n_rep in [1, 5, 10, 20, 30]:
    print("n_rep = ", n_rep, ": ", outCVSVM(X_train, Y_train, 5, n_rep, 0.3))
```

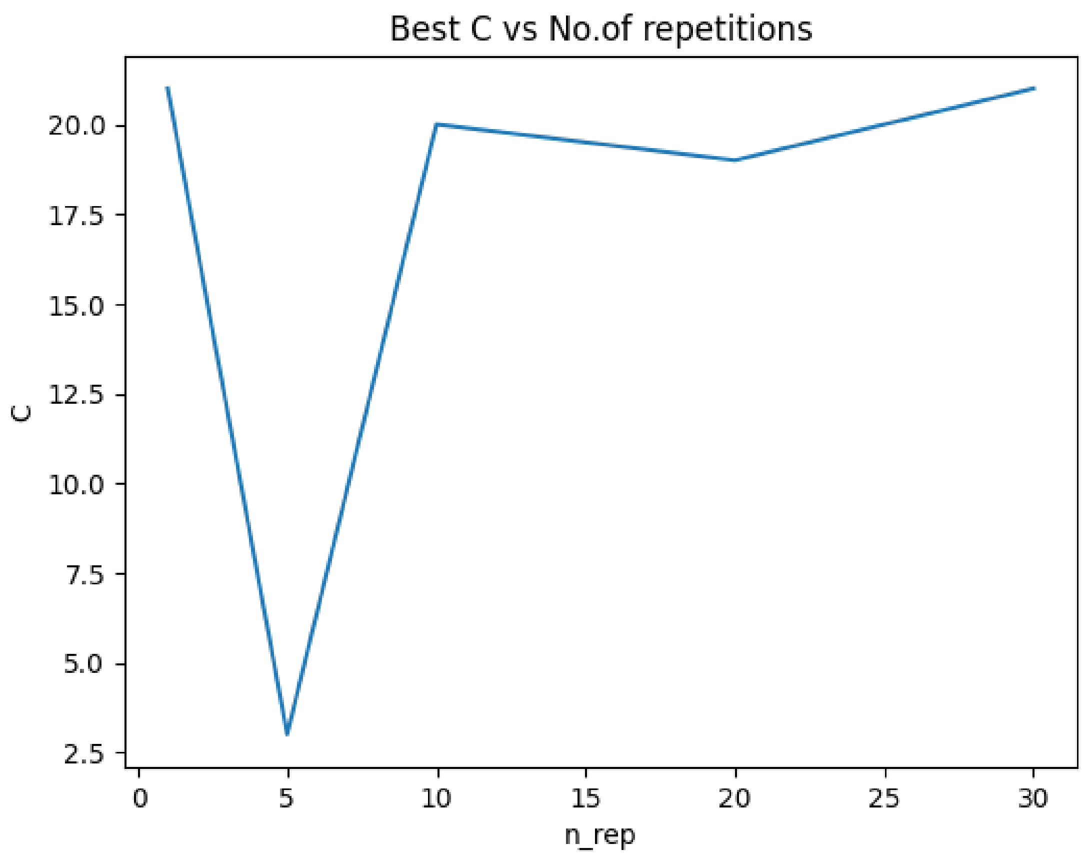
```
n_rep =  1 :  [83.33 75.  ]
n_rep =  5 :  [77.22 78.33]
n_rep =  10 :  [76.11 77.38]
n_rep =  20 :  [74.72 78.51]
n_rep =  30 :  [73.52 78.49]
```

[ ]:
```
best_C_nrep = []
for n_rep in [1, 5, 10, 20, 30]:
    # print("rho = ", rho, ": ", outCVSVM(X_train, Y_train, 5, 10, rho))
    aaa = []
    for C in range(1, 23):
        svmout = outCVSVM(X_train, Y_train, C, n_rep, 0.3)
        aaa.append(svmout)
    best_C_nrep.append(np.argmax([x[0] for x in aaa]))
    # print('N_rep = ' + str(rho) + ': k = ', np.argmax([x[0] for x in␣
 ↪aaa])*2+1, 'is the best k for hold-out')
```

```
    print('For = ' + str(n_rep) + ' repetitions: C =', np.argmax([x[0] for x in␣
↪aaa]), 'is the best')
```

```
For = 1 repetitions: C = 14 is the best
For = 5 repetitions: C = 3 is the best
For = 10 repetitions: C = 21 is the best
For = 20 repetitions: C = 7 is the best
For = 30 repetitions: C = 7 is the best
```

[ ]:
```
#k vs n_rep
plt.plot([1, 5, 10, 20, 30], best_C)
plt.xlabel('n_rep')
plt.ylabel('C')
plt.title('Best C vs No.of repetitions')
plt.show()
```



(c) Make a comparison of the performance of the soft-margin SVM with best value of C and kNN with best k (for the current modified dataset), that is, which one performs better.

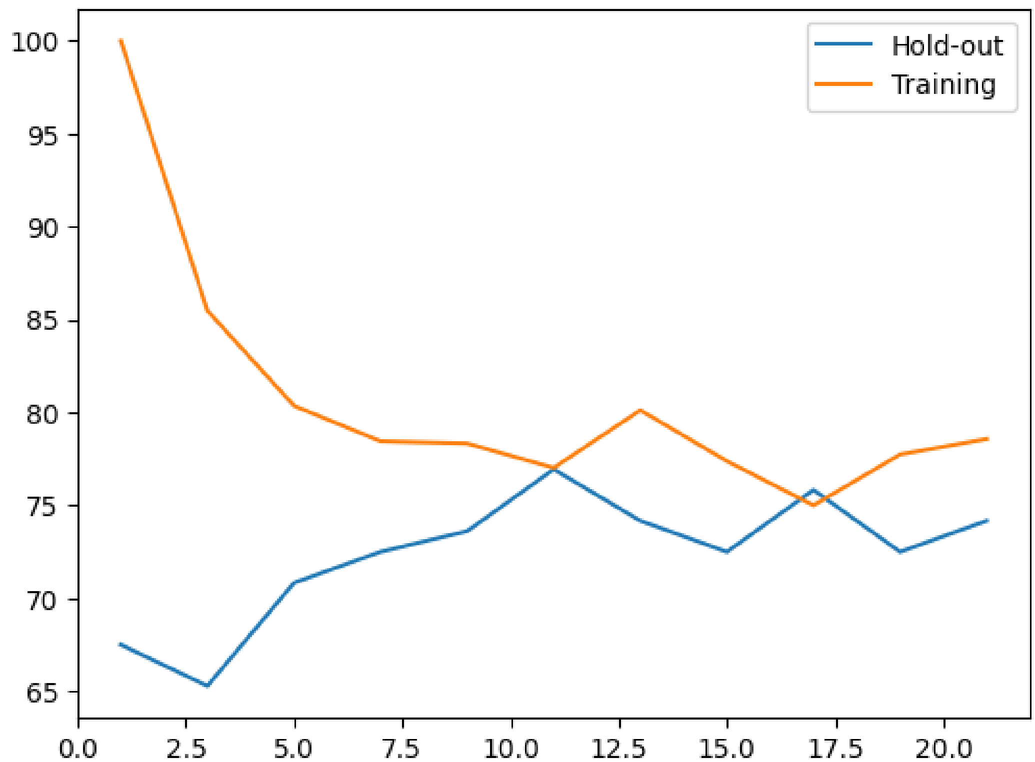
[ ]:
```python
def kNNClassify(X_train, Y_train, X_test, n_neighbors):

    distances = np.sqrt(np.sum((X_test[:, np.newaxis, :] - X_train[np.newaxis, :
 ↪, :])**2, axis=2))

    kNN_loc = np.argsort(distances, axis=1)[:, :n_neighbors]

    k_NN_labels = Y_train[kNN_loc]

    Y_pred = np.array([np.argmax(np.bincount(x+1)) for x in k_NN_labels]) # +1␣
 ↪as bincount needs non-negative integers


    return Y_pred-1
```

[ ]:
```python
hold_out_rho = 0.3
n_rep = 10
def outCVkNN(X, Y, k, n_rep, hold_out_rho):
    n = len(X)
    n_hold_out = int(n*hold_out_rho)
    hold_out_accuracy = np.zeros(n_rep)
    train_accuracy = np.zeros(n_rep)
    for i in range(n_rep):
        shuffle_idx = np.random.permutation(n)
        X_train = X[shuffle_idx]
        Y_train = Y[shuffle_idx]
        X_hold_out = X_train[:n_hold_out]
        Y_hold_out = Y_train[:n_hold_out]
        X_train = X_train[n_hold_out:]
        Y_train = Y_train[n_hold_out:]
        Y_pred_hold_out = kNNClassify(X_train, Y_train, X_hold_out, k)
        hold_out_accuracy[i] = np.count_nonzero(Y_hold_out == Y_pred_hold_out)/
 ↪len(Y_hold_out)
        Y_pred_train = kNNClassify(X_train, Y_train, X_train, k)
        train_accuracy[i] = np.count_nonzero(Y_train == Y_pred_train)/
 ↪len(Y_train)


    return np.round([np.mean(hold_out_accuracy)*100, np.
 ↪mean(train_accuracy)*100], 2)
```

[ ]:
```python
aaa = []
for k in range(1, 23, 2):
    knnout = outCVkNN(X_train, Y_train, k, n_rep, hold_out_rho)
    print("k = ", k, ": ", knnout)
    aaa.append(knnout)
```

```
k =  1 :  [ 67.5 100. ]
k =  3 :  [65.28 85.48]
k =  5 :  [70.83 80.36]
k =  7 :  [72.5  78.45]
k =  9 :  [73.61 78.33]
k =  11 :  [76.94 77.02]
k =  13 :  [74.17 80.12]
k =  15 :  [72.5  77.38]
k =  17 :  [75.83 75.  ]
k =  19 :  [72.5  77.74]
k =  21 :  [74.17 78.57]
```

[ ]:
```
plt.plot(range(1, 23, 2), [x[0] for x in aaa], label="Hold-out")
plt.plot(range(1, 23, 2), [x[1] for x in aaa], label="Training")
plt.legend()
plt.show()
```



[ ]:
```
k_best = np.argmax([x[0] for x in aaa])*2+1
print('k =', k_best, 'is the best k for hold-out')
```

```
k = 11 is the best k for hold-out
```

[ ]:
```python
Y_pred_kNN = kNNClassify(X_train, Y_train, X_test, k_best)
kNN_accuracy = np.count_nonzero(Y_pred_kNN == Y_test)*100 / Y_test.shape[0]
```

[ ]:
```python
params = soft_margin_SVM_fit(X_train, Y_train, C = C_best)
params = params.x
Y_pred_SVM = np.sign(np.dot(X_test, params[:-1]) - params[-1])
SVM_accuracy = np.count_nonzero(Y_pred_SVM == Y_test)*100 / Y_test.shape[0]
```

[ ]:
```python
print('kNN accuracy: ', kNN_accuracy)
print('SVM accuracy: ', SVM_accuracy)

if kNN_accuracy > SVM_accuracy:
    print('kNN is better')

else:
    print('SVM is better')
```

```
kNN accuracy:  78.375
SVM accuracy:  80.25
SVM is better
```