Treci domaci zadatak iz predmeta masinsko ucenje

Dragana Ninkovic 2023/3010

Unos podataka i predprocesiranje

In [1]:
```
# Import libararies
import numpy as np;
import matplotlib.pyplot as plt;
import cvxopt
```

In [18]:
```
# Read data
data = np.loadtxt("C:/Users/Dragana/Downloads/mu-d3/svmData.csv", delimiter
```

In [19]:
```
#
seed_value = 30
np.random.seed(seed_value)
np.random.shuffle(data)
print(data.shape)
```

```
(100, 3)
```

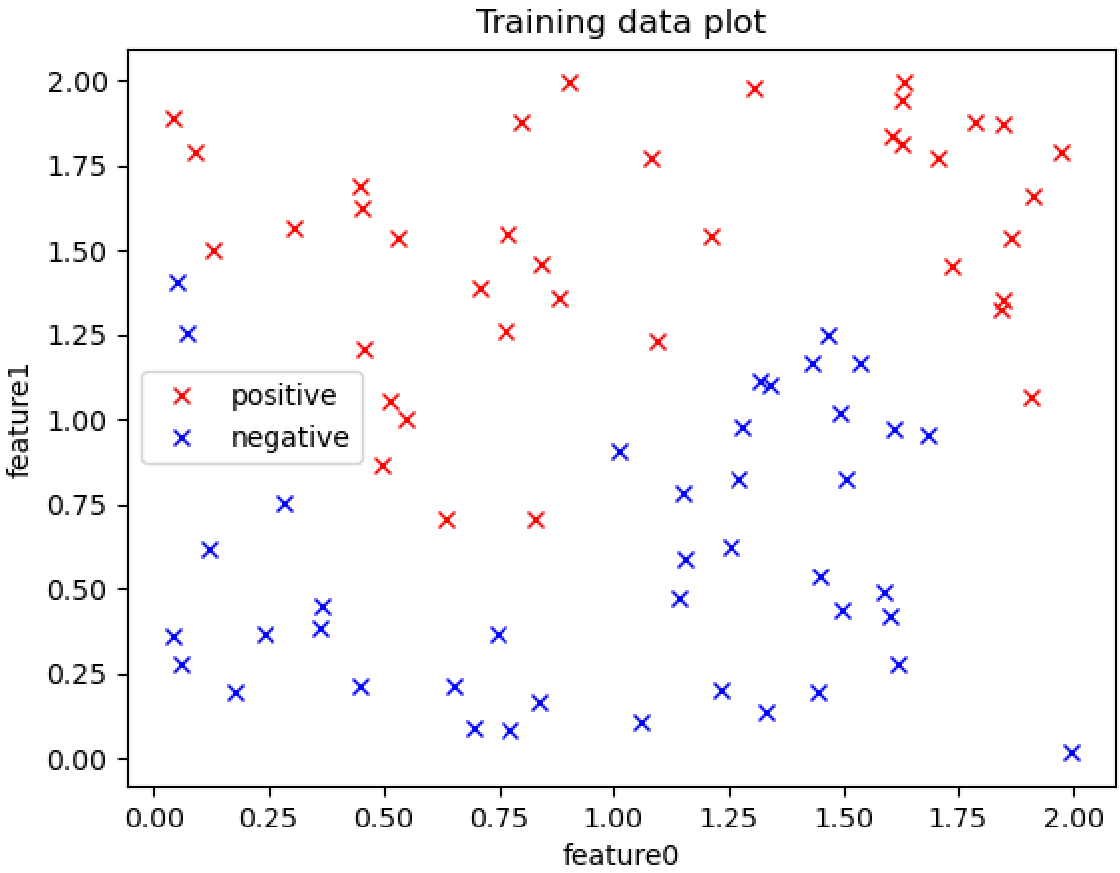
In [20]:
```
# Split on features and labels
X = data[:,0:2];
y = data[:,2];
```

In [21]:
```
# Train test split
train_size = int(0.8*X.shape[0]);
X_train = X[0:train_size,:];
y_train = y[0:train_size];
X_test = X[train_size:,:];
y_test = y[train_size:];
```

In [22]:
```
# Calculate standardization parameters on training set
X_mean = np.mean(X_train, axis = 0);
X_std = np.std(X_train, axis = 0);

X_train_norm = (X_train - X_mean)/X_std;
X_test_norm = (X_test - X_mean)/X_std;
```

In [23]:
```
# Plot training data
plt.figure()
plt.plot(X_train[y_train == 1,0],X_train[y_train == 1,1], 'rx')
plt.plot(X_train[y_train == -1,0],X_train[y_train == -1,1], 'bx')
plt.legend(['positive', 'negative']);
plt.xlabel('feature0')
plt.ylabel('feature1');
plt.title('Training data plot')
```

Out[23]:
```
Text(0.5, 1.0, 'Training data plot')
```



Mozemo videti da ulazne podatke nije moguce odvojiti linearnom granicom i da ocekujemo da ce linearni klasifikator imati vecu gresku od nelinearnog. Takodje, samim tim je neophodno koristiti dozvole ulaska u losu oblast. S druge strane, podaci su dovoljno separabilni da bi trebalo da mogu da se odvoje nekim nelinearnim klasifikatorom ne prevelikog reda.

Unakrsna validacija

Za pronalazenje hiper parametara koriscena je ista funkcija za unaksnu validaciju kao u prvom domacem zadatku.

In [24]:

```python
# Cross validation function for linear SVM
def cross_validation(X,y,num_folds,fold_size,C_values):
    validation_loss_mean = []
    validation_loss_std = []
    train_loss_mean = []
    train_loss_std = []

    for C in C_values:
        current_train_loss = []
        current_validation_loss = []

        for fold in range(num_folds):
            # Split data into training and validation set
            start = fold * fold_size
            end = (fold + 1) * fold_size

            X_validation = X[start:end]
            y_validation = y[start:end].reshape(X_validation.shape[0],1)

            X_train = np.concatenate((X[:start], X[end:]), axis=0)
            y_train = np.concatenate((y[:start], y[end:])).reshape(X_train.

            # Calculate statistic of X_train
            X_mean = np.mean(X_train, axis = 0).reshape(1,X_train.shape[1])
            X_std = np.std(X_train, axis = 0).reshape(1,X_train.shape[1])

            # Standardization
            X_train = (X_train - X_mean)/X_std
            X_validation = (X_validation - X_mean)/X_std

            solution = SVM_primal(X_train, y_train,C);

            b = np.array(solution[0]);
            w = np.array(solution[1:3]);
            psi = np.array(solution[3:]);

            y_validation_pred = linear_predict(X_validation,w,b);
            y_train_pred = linear_predict(X_train,w,b);

            # Keep loss results
            current_validation_loss.append(hinge_loss(y_validation_pred,y_va
            current_train_loss.append(hinge_loss(y_train_pred,y_train))


        # Keep statistics of loss results for this iteration
        validation_loss_mean.append(np.mean(current_validation_loss))
        validation_loss_std.append(np.std(current_validation_loss))
        train_loss_mean.append(np.mean(current_train_loss))
        train_loss_std.append(np.std(current_train_loss))

    # Convert lists to numpy arrays
    validation_loss_mean = np.array(validation_loss_mean)
    validation_loss_std = np.array(validation_loss_std)
    train_loss_mean = np.array(train_loss_mean)
    train_loss_std = np.array(train_loss_std)

    return (validation_loss_mean,validation_loss_std,train_loss_mean,train_
```

Primalni problem i linearan kernel

Funkcija cvxopt za kvadratno programiranje prihvata jednacine u sledecoj formi:

$$min(\frac{1}{2}x^T Px + q^T x)$$
$$Gx \preceq h$$
$$Ax = b$$

, a nas vektor parametara je

$$\begin{bmatrix} b \\ w_0 \\ w_1 \\ \psi_1 \\ \dots \\ \psi_m \end{bmatrix}$$

1. Racunanje matrica P i q:

Nasa kriterijmska funkcija je

$$min(\frac{1}{2}||w||^2 + \sum_1^m \psi_i)$$

. Kako u kvadratnom delu ucestvuje samo vektor w, matrica P ce biti kvadratna matrica ciji su svi elementi nula sem elemenata na dijagonali koji odgovaraju w_0 iw_1. U linearnom clanu ucestvuju samo parametri ψ_i pa ce q biti vektor cija su prva tri elementa nula i ostalo jedinice.

2. Racunanje matrica G i h:

Uslovi pod kojima racunamo kriterijum su da je $1 - \hat{\gamma}^{(i)} - \psi_i \leq 0$ tj. da je $-y^{(i)} w^T (x^{(i)})^T - y^{(i)} b - \psi_i \leq -1$ i da su svi ψ_i pozitivni. Iz ovoga zakljucujemo da ce prvih m redova matrice G koji odgovaraju prvih m nejednakosti imati vektor -y kao prvu kolonu vektor $-y.T * X$ kao druge dve kolone i negiranu jedinicnu matricu kao ostale elemente. Poslednjih m redova matrice G ce ciniti matrica koja ima tri reda jednaka nuli a zatim negiranu jedinicnu matricu. Matrica h kao prvih m elemenata ima -1 a poslednjih m 0.

3. Racunanje matrica A i b

S obzirom da nemamo uslove tipa jednakosti ove dve matrice nam nisu potrebne.

In [25]:
```
# Implementing SVM for primal problem
def SVM_primal(X,y,C):
    n_samples, n_features = X.shape
    P = np.zeros((3+n_samples, 3+n_samples));

    P[1,1] = 1;
    P[2,2] = 1;
    P = cvxopt.matrix(P);

    q = np.ones((n_samples+3,1))*C
    q[0:3] = 0;
    q = cvxopt.matrix(q);

    G_high = np.concatenate((y,y*X, np.eye(n_samples)), axis = 1);
    G_low = np.concatenate((np.zeros((n_samples,3)), np.eye(n_samples)), ax
    G = np.concatenate((G_high, G_low), axis = 0)
    G = cvxopt.matrix(-G);

    h_high = np.ones((n_samples,1))*-1
    h_low = np.zeros((n_samples,1))
    h = np.concatenate((h_high,h_low),axis = 0);
    h = cvxopt.matrix(h);

    return  cvxopt.solvers.qp(P, q, G, h)['x']
```

Za funkciju gubitka koristi se Hinge loss koji je jednak $max(1-\hat{\gamma},0)$ u slucaju linearnog klasifikatora, u slucaju nelinearnog klasifikatora mozemo oduzeti od 1 umnozak prave i prediktovane vrednosti.

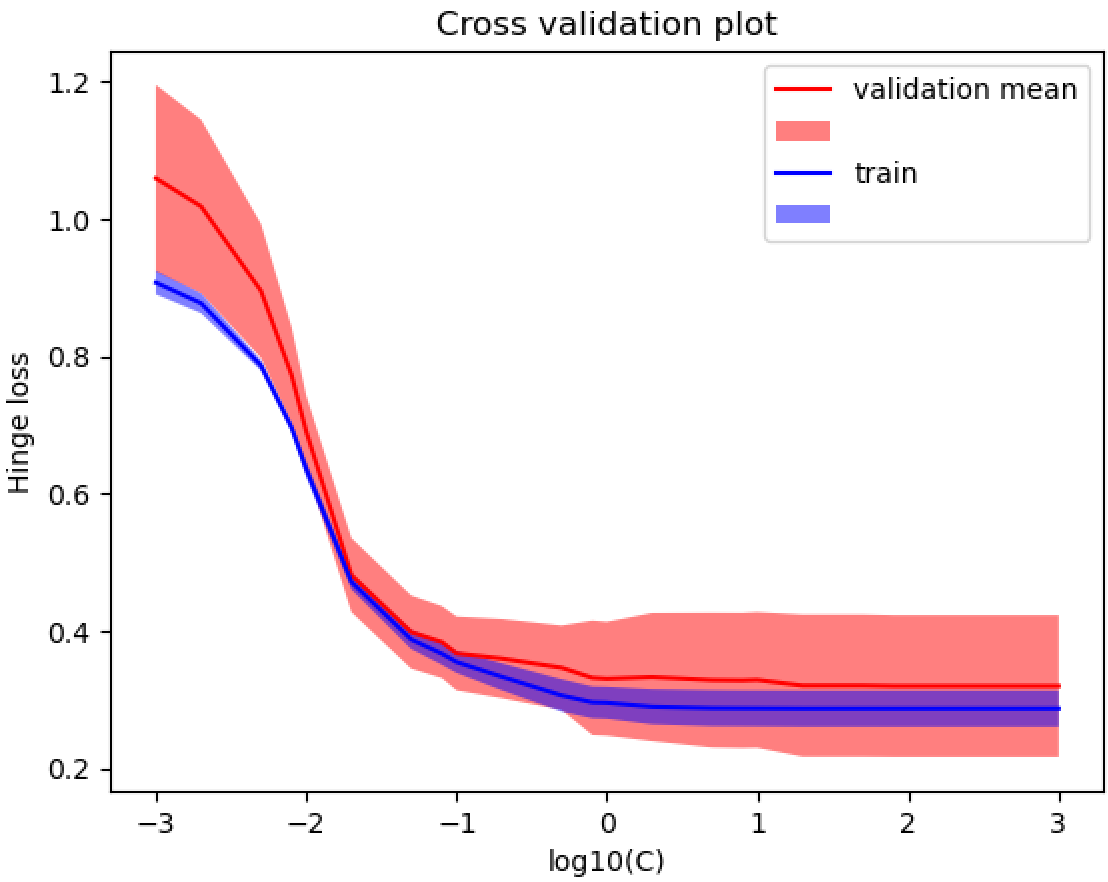
In [26]:
```
# Hinge loss function
def linear_predict(x,w,b):
    return (x@w+b);
def hinge_loss(y_pred, y):
    gamma_hat_i = y*y_pred
    loss = 1-gamma_hat_i;
    loss[loss<0] = 0;
    return  np.sum(loss)/len(y)
```

Biranje hiperparametra C

In [27]:
```
n_samples, n_features = X_train.shape
num_folds = 5
fold_size = n_samples // num_folds
C = [0.001,0.002,0.005,0.008,0.01,0.02,0.05,0.08,0.1,0.2,0.5,0.8,1,2,5,8,10
y_train = y_train.reshape(n_samples,1)
(validation_rmse_mean,validation_rmse_std,train_rmse_mean,train_rmse_std) =
```

```
     pcost       dcost       gap    pres   dres
 0:  3.1490e-01  1.2612e+01  4e+02  3e+00  1e+02
 1:  3.9303e-01 -8.7197e+00  9e+00  6e-02  2e+00
 2:  1.6226e-01 -5.7784e-01  7e-01  4e-03  2e-01
 3:  1.2833e-01  5.2997e-02  8e-02  7e-16  2e-17
 4:  6.4173e-02  5.6379e-02  8e-03  4e-16  1e-17
 5:  5.8028e-02  5.6813e-02  1e-03  3e-16  4e-17
 6:  5.7982e-02  5.6941e-02  1e-03  3e-16  4e-17
 7:  5.7288e-02  5.7103e-02  2e-04  2e-16  3e-17
 8:  5.7188e-02  5.7159e-02  3e-05  3e-16  1e-17
 9:  5.7173e-02  5.7169e-02  4e-06  3e-16  3e-17
10:  5.7171e-02  5.7171e-02  7e-08  3e-16  1e-16
Optimal solution found.
     pcost       dcost       gap    pres   dres
 0:  2.9682e-01  1.3487e+01  4e+02  3e+00  9e+01
 1:  3.7573e-01 -7.5634e+00  8e+00  5e-02  2e+00
 2:  1.5697e-01 -4.6072e-01  6e-01  4e-03  1e-01
 3:  1.2250e-01  5.4245e-02  7e-02  6e-16  6e-17
 4:  6.4849e-02  5.8380e-02  6e-03  4e-16  1e-17
```

In [28]:
```
plt.figure()
plt.plot(np.log10(C),validation_rmse_mean,c='r')
plt.fill_between(np.log10(C),validation_rmse_mean-validation_rmse_std,valid
plt.plot(np.log10(C),train_rmse_mean, c= 'b')
plt.fill_between(np.log10(C),train_rmse_mean-train_rmse_std,train_rmse_mean
plt.xlabel('log10(C)')
plt.ylabel('Hinge loss')
plt.legend(['validation mean', '','train',''])
plt.title("Cross validation plot")
plt.show()
```

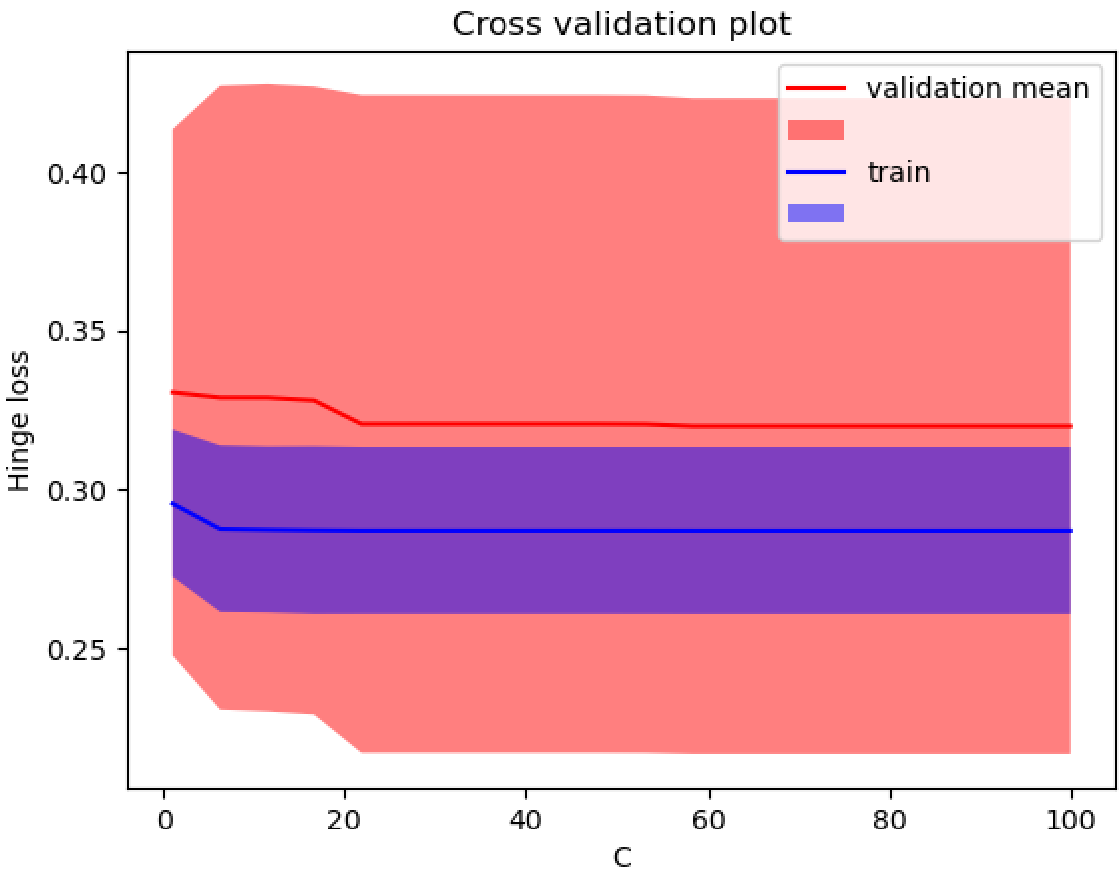


Mozemo primetiti da i validaciona i obucavajuca kriva dostizu minimum na opsegu 1 do 100. Ovaj opseg je dodatno posmatran u cilju odredjivanja tacnog parametra C

In [32]:
```
C = list(np.linspace(1,100,num = 20))

(validation_rmse_mean,validation_rmse_std,train_rmse_mean,train_rmse_std) =
```

```
     pcost       dcost       gap    pres   dres
 0: -3.7223e+01  1.2934e+02  6e+02  4e+00  1e+01
 1:  6.3719e+01 -1.0401e+01  1e+02  4e-01  1e+00
 2:  2.4982e+01  1.4480e+01  1e+01  4e-02  1e-01
 3:  2.0938e+01  1.8358e+01  3e+00  7e-03  2e-02
 4:  2.0124e+01  1.9477e+01  7e-01  1e-03  4e-03
 5:  1.9843e+01  1.9734e+01  1e-01  4e-16  1e-14
 6:  1.9793e+01  1.9786e+01  7e-03  4e-16  2e-15
 7:  1.9789e+01  1.9789e+01  8e-05  4e-16  8e-15
 8:  1.9789e+01  1.9789e+01  8e-07  4e-16  4e-15
Optimal solution found.
     pcost       dcost       gap    pres   dres
 0: -3.5582e+01  1.2437e+02  5e+02  4e+00  1e+01
 1:  6.5373e+01  5.3668e+00  7e+01  2e-01  5e-01
 2:  2.8018e+01  1.8518e+01  1e+01  3e-02  7e-02
 3:  2.4120e+01  2.1607e+01  3e+00  6e-03  2e-02
 4:  2.3162e+01  2.2561e+01  7e-01  1e-03  3e-03
 5:  2.2858e+01  2.2814e+01  5e-02  8e-05  2e-04
 6:  2.2835e+01  2.2835e+01  5e-04  8e-07  2e-06
```

In [34]:
```
plt.figure()
plt.plot(C,validation_rmse_mean,c='r')
plt.fill_between(C,validation_rmse_mean-validation_rmse_std,validation_rmse_
plt.plot(C,train_rmse_mean, c= 'b')
plt.fill_between(C,train_rmse_mean-train_rmse_std,train_rmse_mean+train_rms
plt.legend(['validation mean', '','train',''])
plt.xlabel('C')
plt.ylabel('Hinge loss')
plt.title("Cross validation plot")
plt.show()
```



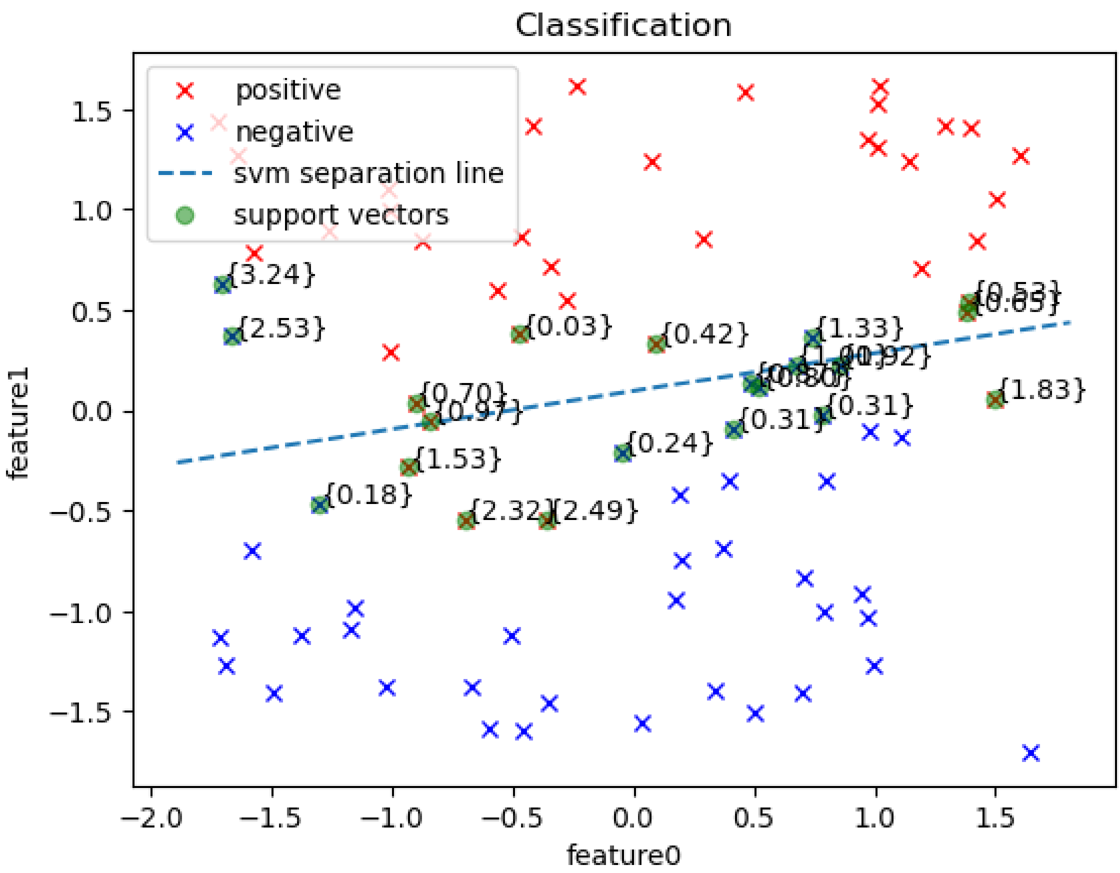
In [77]:

```
C_opt = 20
solution = SVM_primal(X_train_norm, y_train,C_opt);

b = np.array(solution[0,0]).reshape(1,1);
w = np.array(solution[1:3,0]).reshape(2,1);
psi = np.array(solution[3:]);
```

```
     pcost       dcost       gap    pres   dres
 0: -2.5189e+04  9.8239e+03  3e+04  4e+01  2e+00
 1:  2.5371e+03  7.7067e+01  3e+03  2e-01  1e-02
 2:  7.0587e+02  3.0724e+02  4e+02  3e-02  2e-03
 3:  6.0259e+02  3.8628e+02  2e+02  1e-02  6e-04
 4:  5.2354e+02  4.2158e+02  1e+02  4e-03  2e-04
 5:  4.9732e+02  4.4618e+02  5e+01  1e-03  8e-05
 6:  4.8471e+02  4.5394e+02  3e+01  4e-04  2e-05
 7:  4.6888e+02  4.6693e+02  2e+00  9e-06  5e-07
 8:  4.6778e+02  4.6773e+02  5e-02  2e-07  1e-08
 9:  4.6775e+02  4.6775e+02  5e-04  2e-09  1e-10
10:  4.6775e+02  4.6775e+02  5e-06  2e-11  1e-12
Optimal solution found.
```

In [85]:
```python
plt.figure()
label_1 = ((y_train == 1).T)[0]
label_minus_1 = ((y_train == -1).T)[0]
plt.plot(X_train_norm[label_1,0],X_train_norm[label_1,1], 'rx')
plt.plot(X_train_norm[label_minus_1,0],X_train_norm[label_minus_1,1], 'bx')
xl, xr = plt.xlim()
x_axis = np.array([xl, xr]).reshape(2,1)
y_svm = -(b + w[0] * x_axis) / w[ 1]
plt.plot(x_axis, y_svm, '--', label='SVM')

for i, txt in enumerate(psi):
    if(txt > 1e-5):
       plt.annotate('{%.2f}'%(txt), (X_train_norm[i,0], X_train_norm[i,1]))
       plt.plot(X_train_norm[i,0], X_train_norm[i,1],'go', alpha = 0.5)

plt.legend(['positive', 'negative','svm separation line','support vectors']
plt.xlabel('feature0')
plt.ylabel('feature1');
plt.title('Classification')
```

Out[85]:
```
Text(0.5, 1.0, 'Classification')
```



Na slici su zelenom bojom naznaceni potporni vektori i njihove vrednosti su zapisane pored. Mozemo uociti da su za potporne vektore uzeti oni elementi koji su blizu separacione linije i oni koji su pogresno klasifikovani. Takodje, oni koji su sa pogresne strane klasifikacione linije imaju vrednosti ψ vece od 1 dok oni koji su sa prave imaju vrednosti manje od 1.

In [80]:
```
y_test = y_test.reshape(X_test.shape[0],1)
y_pred_train = np.sign(linear_predict(X_train_norm,w,b))
acc_train = np.sum(y_pred_train == y_train.reshape(len(y_train),1))/len(y_tr
y_pred_test = np.sign(linear_predict(X_test_norm,w,b));
acc_test = np.sum(y_pred_test == y_test.reshape(len(y_test),1))/len(y_test)
print(acc_train)
print(acc_test)
```

```
0.9
0.85
```

Krajnja tacnost ovog modela je 90% na obucavajucem skupu i 85% na testirajucem skupu.

In [81]:
```
train_loss = hinge_loss(y_pred_train, y_train)
test_loss = hinge_loss(y_pred_test, y_test)
```

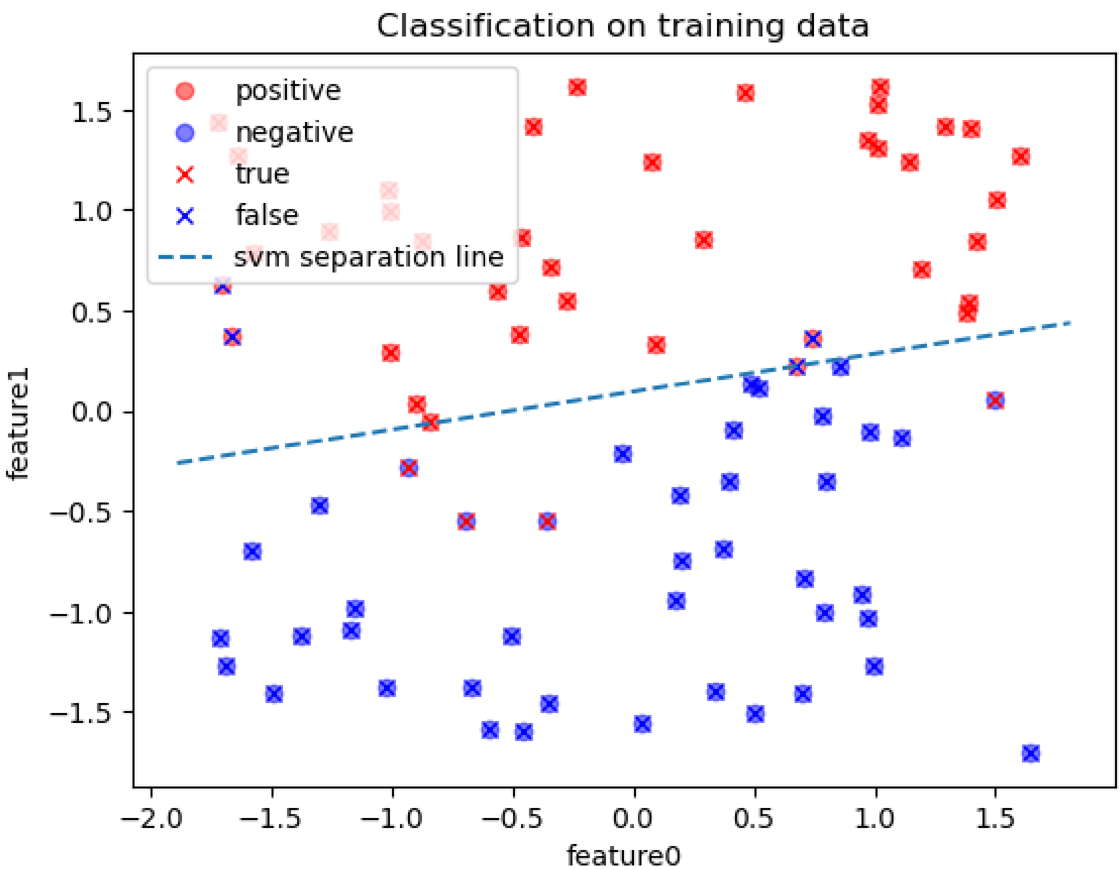
In [82]:
```
print(train_loss)
print(test_loss)
```

```
0.2
0.3
```

Vidimo da su i velicine gubitaka slicne, ali je nesto veci za testirajuci skup.

In [84]:
```
plt.figure()
plt.plot(X_train_norm[y_pred_train[:,0] == 1,0],X_train_norm[y_pred_train[:
plt.plot(X_train_norm[y_pred_train[:,0]  == -1,0],X_train_norm[y_pred_train
plt.plot(X_train_norm[label_1,0],X_train_norm[label_1,1], 'rx')
plt.plot(X_train_norm[label_minus_1,0],X_train_norm[label_minus_1,1], 'bx')
xl, xr = plt.xlim()
x_axis = np.array([xl, xr]).reshape(2,1)
y_svm = -(b + w[0] * x_axis) / w[ 1]
plt.plot(x_axis, y_svm, '--', label='SVM')

plt.legend(['positive', 'negative','true','false','svm separation line']);
plt.xlabel('feature0')
plt.ylabel('feature1');
plt.title('Classification on training data')
```

Out[84]:
```
Text(0.5, 1.0, 'Classification on training data')
```

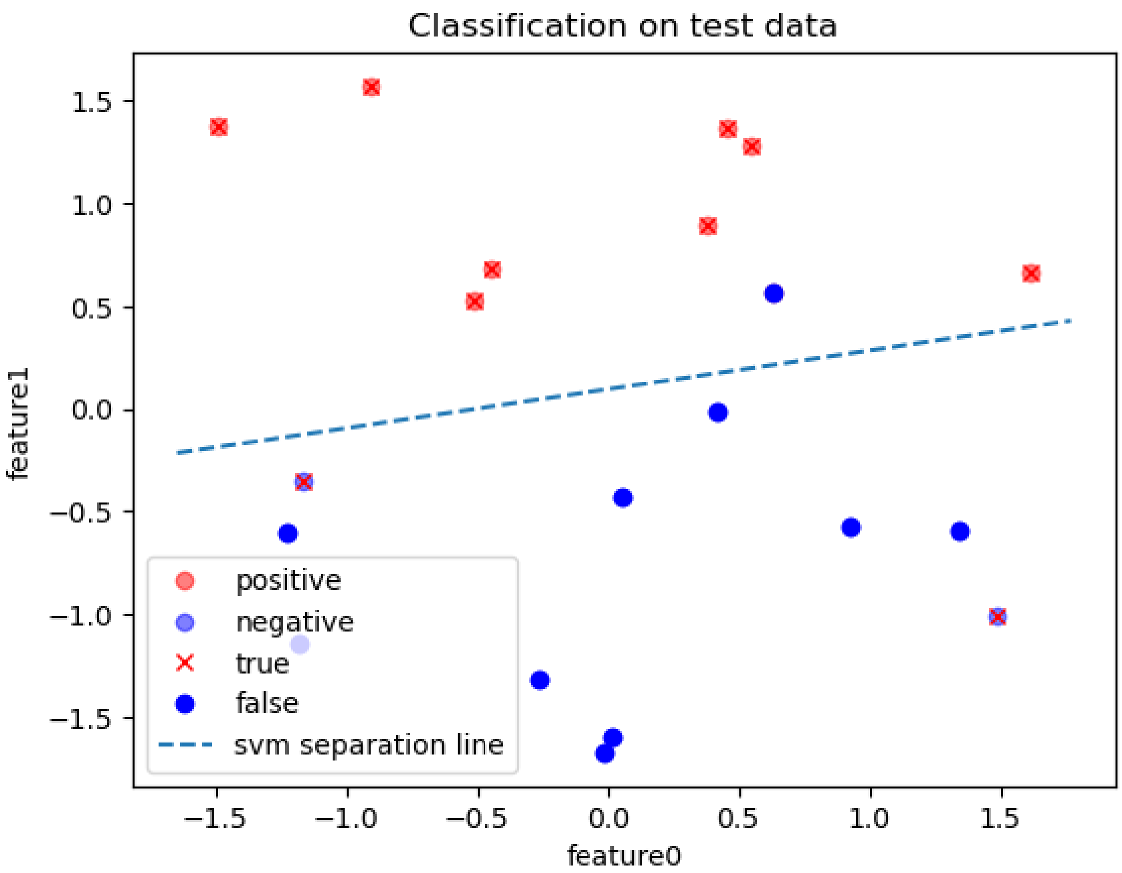


Na prethodnoj slici prikazane su predikcije na trenirajucem skupu a na sledecoj na testirajucem.

In [74]:
```
label_1_test = ((y_test == 1).T)[0]
label_minus_1_test = ((y_test == -1).T)[0]
```

In [75]:
```
plt.figure()
plt.plot(X_test_norm[y_pred_test[:,0] == 1,0],X_test_norm[y_pred_test[:,0]
plt.plot(X_test_norm[y_pred_test[:,0]  == -1,0],X_test_norm[y_pred_test[:,0
plt.plot(X_test_norm[label_1_test,0],X_test_norm[label_1_test,1], 'rx')
plt.plot(X_test_norm[label_minus_1_test,0],X_test_norm[label_minus_1_test,1
xl, xr = plt.xlim()
x_axis = np.array([xl, xr]).reshape(2,1)
y_svm = -(b + w[0] * x_axis) / w[ 1]
plt.plot(x_axis, y_svm, '--', label='SVM')

plt.legend(['positive', 'negative','true','false','svm separation line']);
plt.xlabel('feature0')
plt.ylabel('feature1');
plt.title('Classification on test data')
```

Out[75]: Text(0.5, 1.0, 'Classification on test data')



Dualni problem i nelinearan kernel

Podsetimo se forme koju prima nasa funkcija:

$$min(\frac{1}{2}x^T Px + q^T x)$$
$$Gx \preceq h$$
$$Ax = b$$

, ovog puta nas vektor parametara je

$$\begin{bmatrix} \alpha_1 \\ \dots \\ \alpha_m \end{bmatrix}$$

1. Racunanje matrica P i q:

Nasa kriterijmska funkcija je

$$max(-\frac{1}{2}\sum_{1}^{m}\sum_{1}^{m}\alpha_i\alpha_j y_i y_j K(x_i, x_j) + \sum_{1}^{m}\alpha_i) = min(\frac{1}{2}\sum_{1}^{m}\sum_{1}^{m}\alpha_i\alpha_j y_i y_j K(x_i, x_j) - \sum_{1}^{m}$$

.

Pa je matrica P sada Y*K gde je Y* = *y*y.T a matrica q samo vektor -1.

2. Racunanje matrica G i h:

Jedine nejednakosti koje treba da budu zadovoljene su da je α_i pozitivno i manje od C sto su znacajno jednostavniji uslovi nego malo pre. Iz tog razloga matrica G ce se sastojati od jedinicne i negativne jedinicne matrice dok ce h biti vektor cijih je prvih m elemenata C a poslednjih m 0.

3. Racunanje matrica A i b

U ovom slucaju nam jesu potrebne i matrice A i b jer imamo uslov u obliku jednakosti da je $\sum_1^m y_i\alpha_i = 0$

Iz toga mozemo zakljuciti da je A = y.T a b = 0.

S obzirom na ispunjenost KKT uslova resenje dualnog problema je ekvivalentno resenju primalnog i bice mnogo jednostavnije resiti ga zbog jednostavnijih matrica iako je njegova forma idalje kvadratna.

In [86]:
```
# SVM function for dual problem
def SVM_dual(X,y,C, K):
    n_samples, n_features = X.shape

    y = y.reshape(n_samples,1)
    Y = y*(y.T)

    P = K*Y
    P = cvxopt.matrix(P);

    q = -1.0*np.ones((n_samples,1))
    q = cvxopt.matrix(q);

    G = np.concatenate((np.eye(n_samples), -1.0*np.eye(n_samples)), axis = (
    G = cvxopt.matrix(G);

    h_high = np.ones((n_samples,1))*C
    h_low = np.zeros((n_samples,1))
    h = np.concatenate((h_high,h_low),axis = 0);
    h = cvxopt.matrix(h);

    A = y.T
    A = cvxopt.matrix(A);

    b = 0.0
    b = cvxopt.matrix(b);

    return  cvxopt.solvers.qp(P, q, G, h,A,b)['x']
```

Kerneli

Dva najcesce koriscena kernela su gausov i polinomijalni kernel. Gausov kernel je slican polinomijalnom koji ima beskonacan stepen. S obzirom na izgled podataka deluje da ce polinomijalni kernel biti dovoljan ali bice isprobana oba.

In [87]:
```
# Kernel functions
def gaussian_kernel(X,x, sigma):
    n_samples, n_features = X.shape
    XX = X.reshape((n_samples, n_features,1));
    K = XX - x.T
    K = K*K;
    K = np.sum(K, axis = 1);
    K = np.exp(-K/2/sigma/sigma)
    return K

def polynomial_kernel(X,x,c,d):
    K = np.zeros((n_samples, n_samples))
    K = (c+np.dot(X,x.T))**d
    return K
def predict(K,alpha,y,b):
    y_pred = np.sign(((alpha*y).T)@K+b)
    return y_pred
```

In [88]:
```python
def cross_validation(X,y,num_folds,fold_size,all_params, var_params, var_pa
    validation_loss_mean = []
    validation_loss_std = []
    train_loss_mean = []
    train_loss_std = []

    for param in var_params:
            current_train_loss = []
            current_validation_loss = []
            all_params[var_param_name] = param;

            for fold in range(num_folds):
                # Split data into training and validation set
                start = fold * fold_size
                end = (fold + 1) * fold_size

                X_validation = X[start:end]
                y_validation = y[start:end].reshape(X_validation.shape[0],1

                X_train = np.concatenate((X[:start], X[end:]), axis=0)
                y_train = np.concatenate((y[:start], y[end:])).reshape(X_tr

                # Calculate statistic of X_train
                X_mean = np.mean(X_train, axis = 0)
                X_std = np.std(X_train, axis = 0)

                # Standardization
                X_train = (X_train - X_mean)/X_std
                X_validation = (X_validation - X_mean)/X_std

                if(kernel_type == 'P'):
                        K_train = polynomial_kernel(X_train,X_train,all_para
                        K_validation = polynomial_kernel(X_train,X_validati
                else:
                        K_train = gaussian_kernel(X_train,X_train,all_param
                        K_validation = gaussian_kernel(X_train,X_validation

                C = all_params['C'];
                alpha = SVM_dual(X_train, y_train,C, K_train)
                alpha = np.array(alpha).reshape(X_train.shape[0],1)

                support_id = (np.logical_and((alpha > 1e-5), (alpha<C)).T)[
                support_y = y_train[support_id,:][0]
                b = (1/support_y - np.sum(alpha*y_train*((K_train[:,support_

                y_train_pred = predict(K_train,alpha,y_train,b ).T
                y_validation_pred = predict(K_validation,alpha,y_train,b).T

                # Keep loss results
                current_validation_loss.append(hinge_loss(y_validation_pred
                current_train_loss.append(hinge_loss(y_train_pred,y_train))


            # Keep statistics of loss results for this iteration
            validation_loss_mean.append(np.mean(current_validation_loss))
            validation_loss_std.append(np.std(current_validation_loss))
            train_loss_mean.append(np.mean(current_train_loss))
            train_loss_std.append(np.std(current_train_loss))

    # Convert lists to numpy arrays
    validation_loss_mean = np.array(validation_loss_mean)
```

```python
    validation_loss_std = np.array(validation_loss_std)
    train_loss_mean = np.array(train_loss_mean)
    train_loss_std = np.array(train_loss_std)

    return (validation_loss_mean,validation_loss_std,train_loss_mean,train_l
```

Funkcija za kros validaciju je ista kao malo pre s tim sto umesto jednog parametra sada bira recnik parametara all_params, vrednosti kroz koje treba da prodje parametar koji biramo - var_params i ime parametra koji biramo. Takodje, s obzirom da imamo dva kernela kernel_type je P za polinomijalni kernel a G za gausov.

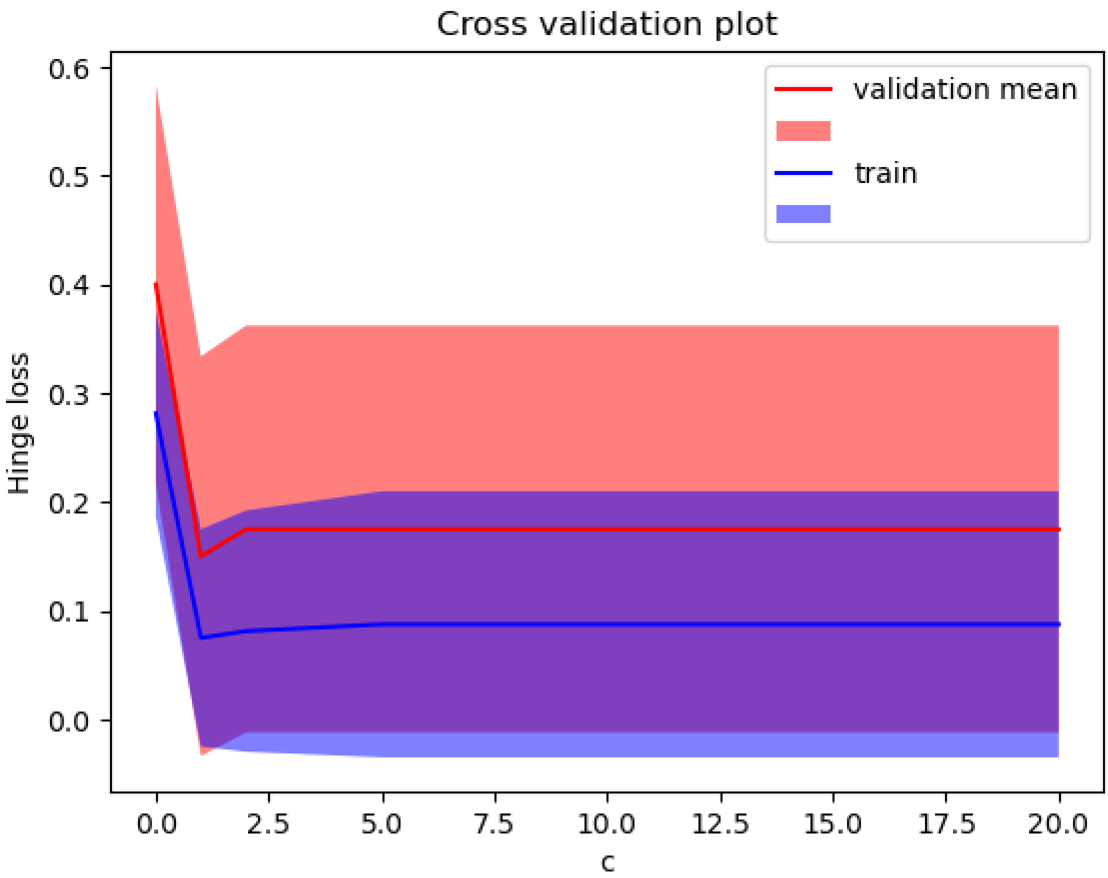
Biranje hiperparametara za polinomijalni kernel

In [89]:
```python
all_params = {'C':20, 'c' : 0, 'd' : 3, 'sigma' : 1 }
var_params = [0,1,2,5,7,10, 20]
var_param_name = 'c'
kernel_type = 'P'
```

In [90]:
```python
(validation_rmse_mean,validation_rmse_std,train_rmse_mean,train_rmse_std) =
```

```
     pcost       dcost       gap    pres   dres
 0: -3.2485e+02 -2.0429e+04  5e+04  7e-01  6e-14
 1: -2.2291e+02 -8.7070e+03  1e+04  1e-01  9e-14
 2: -1.6563e+02 -1.9901e+03  2e+03  2e-02  7e-14
 3: -2.2679e+02 -7.7034e+02  6e+02  6e-03  3e-14
 4: -3.2460e+02 -4.7816e+02  2e+02  7e-04  7e-14
 5: -3.6122e+02 -4.1370e+02  5e+01  1e-04  4e-14
 6: -3.7360e+02 -3.9405e+02  2e+01  4e-05  6e-14
 7: -3.7917e+02 -3.8502e+02  6e+00  7e-06  4e-14
 8: -3.8140e+02 -3.8199e+02  6e-01  5e-07  4e-14
 9: -3.8164e+02 -3.8167e+02  3e-02  1e-14  7e-14
10: -3.8166e+02 -3.8166e+02  3e-04  7e-15  6e-14
Optimal solution found.
     pcost       dcost       gap    pres   dres
 0: -2.8901e+02 -1.5097e+04  3e+04  5e-01  7e-14
 1: -2.1895e+02 -4.8176e+03  6e+03  7e-02  6e-14
 2: -2.1069e+02 -1.0307e+03  9e+02  9e-03  5e-14
 3: -3.1954e+02 -5.5188e+02  2e+02  2e-03  4e-14
 4: -3.8562e+02 -4.6584e+02  8e+01  3e-04  4e-14
```

In [92]:
```
plt.figure()
plt.plot(var_params,validation_rmse_mean,c='r')
plt.fill_between(var_params,validation_rmse_mean-validation_rmse_std,valida
plt.plot(var_params,train_rmse_mean, c= 'b')
plt.fill_between(var_params,train_rmse_mean-train_rmse_std,train_rmse_mean+
plt.legend(['validation mean', '','train',''])
plt.title("Cross validation plot")
plt.xlabel('c')
plt.ylabel('Hinge loss')
plt.show()
```

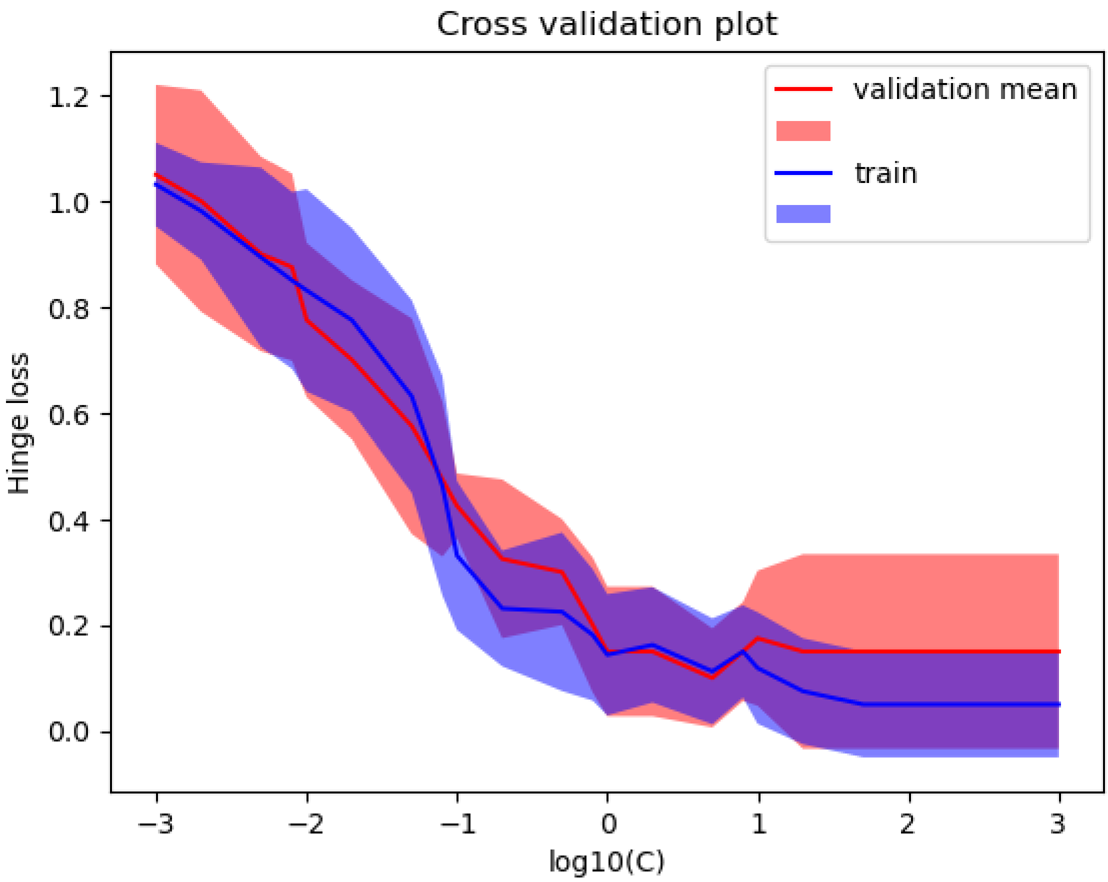


In [93]:
```
all_params = {'C':20, 'c' : 1, 'd' : 3, 'sigma' : 1 }
var_params =  [0.001,0.002,0.005,0.008,0.01,0.02,0.05,0.08,0.1,0.2,0.5,0.8,
var_param_name = 'C'
kernel_type = 'P'
```

In [94]:
```
(validation_rmse_mean,validation_rmse_std,train_rmse_mean,train_rmse_std) =
```

```
     pcost       dcost       gap    pres   dres
 0: -7.9132e+00 -4.2276e-01  3e+02  2e+01  7e-15
 1: -5.5916e-01 -3.4416e-01  9e+00  5e-01  8e-15
 2: -1.1427e-01 -1.4464e-01  9e-01  5e-02  1e-15
 3: -4.3362e-02 -1.1928e-01  8e-02  6e-18  1e-15
 4: -4.6804e-02 -5.7090e-02  1e-02  2e-18  6e-16
 5: -4.8680e-02 -5.2721e-02  4e-03  2e-18  4e-16
 6: -4.9657e-02 -5.0523e-02  9e-04  2e-18  4e-16
 7: -4.9966e-02 -5.0050e-02  8e-05  3e-18  5e-16
 8: -5.0001e-02 -5.0004e-02  3e-06  2e-18  5e-16
 9: -5.0002e-02 -5.0002e-02  1e-07  2e-18  5e-16
Optimal solution found.
     pcost       dcost       gap    pres   dres
 0: -7.6397e+00 -4.7445e-01  3e+02  2e+01  5e-15
 1: -6.7214e-01 -3.5278e-01  1e+01  6e-01  6e-15
 2: -1.1203e-01 -1.4654e-01  9e-01  5e-02  1e-15
 3: -4.4688e-02 -1.1904e-01  7e-02  6e-18  1e-15
 4: -4.8570e-02 -5.5321e-02  7e-03  1e-18  6e-16
 5: -4.9454e-02 -5.4366e-02  5e-03  1e-18  4e-16
```

In [97]:
```
plt.figure()
plt.plot(np.log10(var_params),validation_rmse_mean,c='r')
plt.fill_between(np.log10(var_params),validation_rmse_mean-validation_rmse_s
plt.plot(np.log10(var_params),train_rmse_mean, c= 'b')
plt.fill_between(np.log10(var_params),train_rmse_mean-train_rmse_std,train_r
plt.legend(['validation mean', '','train',''])
plt.title("Cross validation plot")
plt.xlabel('log10(C)')
plt.ylabel('Hinge loss')
plt.show()
```

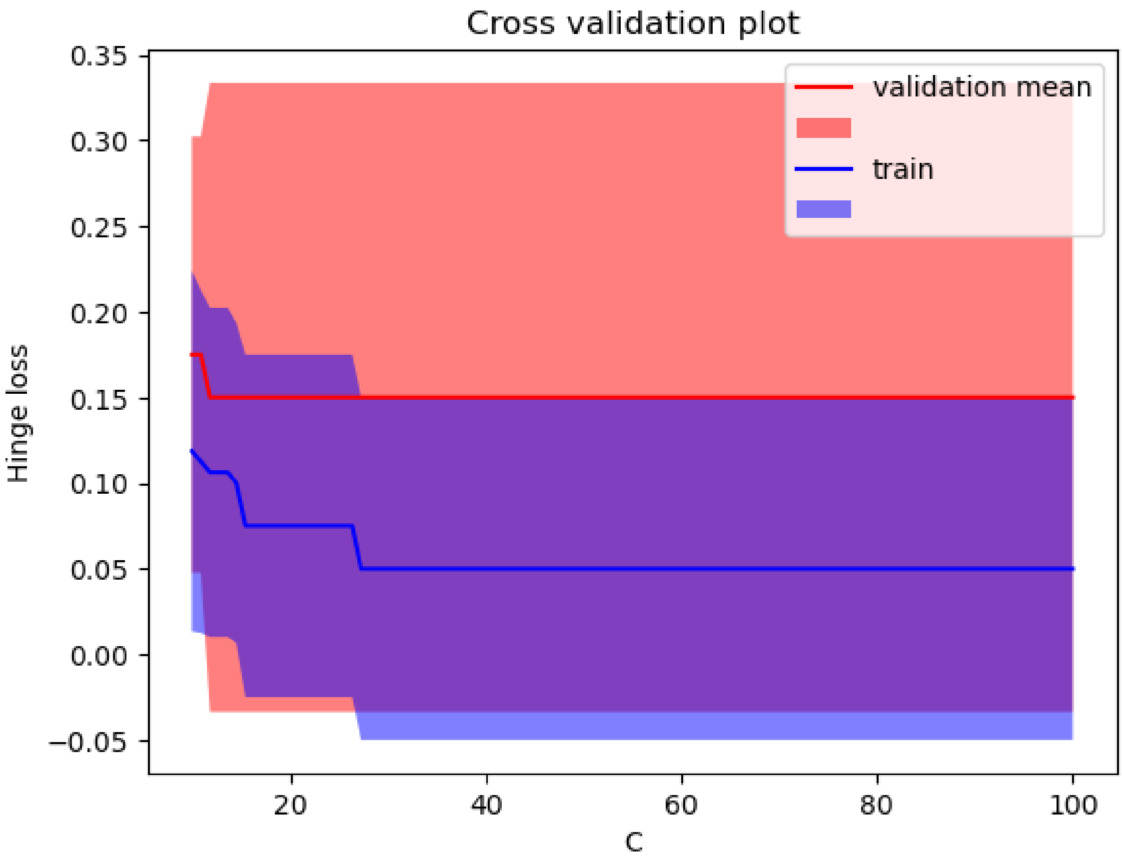


In [98]:
```
all_params = {'C':20, 'c' : 1, 'd' : 3, 'sigma' : 1 }
var_params =  list(np.linspace(10,100,100))
var_param_name = 'C'
kernel_type = 'P'
```

```
In [99]: (validation_rmse_mean,validation_rmse_std,train_rmse_mean,train_rmse_std) =
```

```
     pcost       dcost       gap    pres   dres
 0: -5.2775e+01 -3.7813e+03  9e+03  6e-01  9e-14
 1:  3.9208e+01 -1.1037e+03  2e+03  9e-02  9e-14
 2:  2.3736e+01 -2.2129e+02  3e+02  1e-02  4e-14
 3: -6.2432e+00 -5.5159e+01  5e+01  1e-03  2e-14
 4: -1.6515e+01 -3.0650e+01  2e+01  3e-04  1e-14
 5: -2.1269e+01 -2.3119e+01  2e+00  1e-06  1e-14
 6: -2.1947e+01 -2.2558e+01  6e-01  3e-07  9e-15
 7: -2.2222e+01 -2.2264e+01  4e-02  1e-08  1e-14
 8: -2.2241e+01 -2.2241e+01  5e-04  1e-10  9e-15
 9: -2.2241e+01 -2.2241e+01  5e-06  1e-12  1e-14
Optimal solution found.
     pcost       dcost       gap    pres   dres
 0: -4.3931e+01 -4.6796e+03  1e+04  8e-01  6e-14
 1:  5.4511e+01 -1.7909e+03  3e+03  1e-01  6e-14
 2:  3.9033e+01 -2.4306e+02  4e+02  1e-02  2e-14
 3: -1.4850e+00 -6.3431e+01  7e+01  2e-03  1e-14
 4: -1.5166e+01 -3.1320e+01  2e+01  4e-04  8e-15
 5: -2.0813e+01 -2.3533e+01  3e+00  2e-06  8e-15
```

In [101]:
```
plt.figure()
plt.plot(var_params,validation_rmse_mean,c='r')
plt.fill_between(var_params,validation_rmse_mean-validation_rmse_std,valida
plt.plot(var_params,train_rmse_mean, c= 'b')
plt.fill_between(var_params,train_rmse_mean-train_rmse_std,train_rmse_mean+
plt.legend(['validation mean', '','train',''])
plt.title("Cross validation plot")
plt.xlabel('C')
plt.ylabel('Hinge loss')
plt.show()
```



Da ne bi doslo do preobucavanja uzeto je C = 10

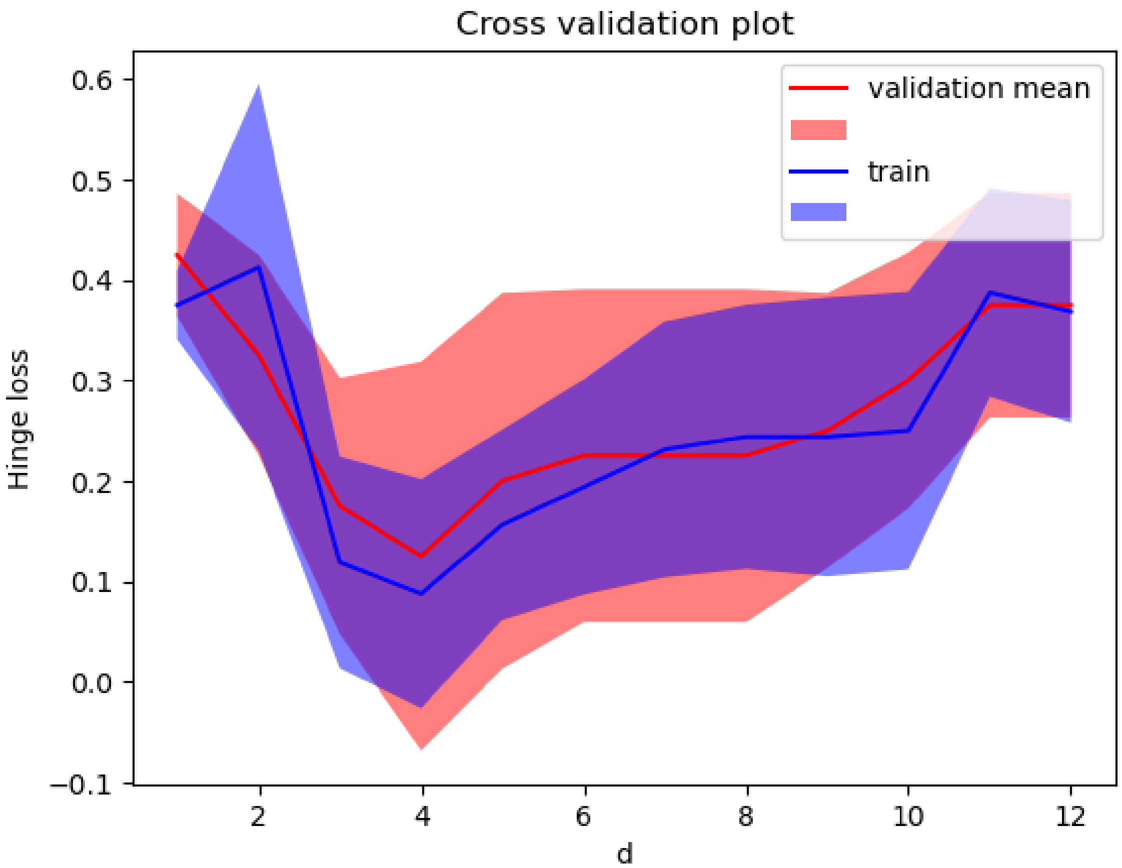
In [102]:
```
all_params = {'C':10, 'c' : 1, 'd' : 3, 'sigma' : 1 }
var_params =  [1,2,3,4,5,6,7,8,9,10,11,12]
var_param_name = 'd'
kernel_type = 'P'
```

In [103]:
```
(validation_rmse_mean,validation_rmse_std,train_rmse_mean,train_rmse_std) =
```

```
     pcost       dcost       gap    pres   dres
 0: -1.0415e+02 -3.9987e+03  8e+03  6e-01  8e-15
 1: -6.6899e+01 -8.5923e+02  8e+02  4e-16  1e-14
 2: -1.2402e+02 -2.7455e+02  2e+02  2e-14  1e-14
 3: -1.5268e+02 -2.1904e+02  7e+01  3e-14  1e-14
 4: -1.6360e+02 -1.9322e+02  3e+01  8e-15  1e-14
 5: -1.7035e+02 -1.8280e+02  1e+01  3e-15  1e-14
 6: -1.7455e+02 -1.7775e+02  3e+00  1e-14  1e-14
 7: -1.7584e+02 -1.7613e+02  3e-01  6e-15  1e-14
 8: -1.7596e+02 -1.7597e+02  9e-03  6e-15  1e-14
 9: -1.7596e+02 -1.7596e+02  9e-05  5e-15  1e-14
Optimal solution found.
     pcost       dcost       gap    pres   dres
 0: -1.1588e+02 -2.9087e+03  5e+03  3e-01  1e-14
 1: -1.1561e+02 -5.3577e+02  5e+02  1e-02  9e-15
 2: -1.5751e+02 -2.8432e+02  1e+02  3e-03  1e-14
 3: -1.8137e+02 -2.5715e+02  8e+01  2e-03  1e-14
 4: -1.9077e+02 -2.4353e+02  5e+01  1e-03  1e-14
 5: -1.9957e+02 -2.2648e+02  3e+01  3e-04  1e-14
```

In [110]:

```
plt.figure()
plt.plot(var_params,validation_rmse_mean,c='r')
plt.fill_between(var_params,validation_rmse_mean-validation_rmse_std,valida
plt.plot(var_params,train_rmse_mean, c= 'b')
plt.fill_between(var_params,train_rmse_mean-train_rmse_std,train_rmse_mean+
plt.legend(['validation mean', '','train',''])
plt.title("Cross validation plot")
plt.xlabel('d')
plt.ylabel('Hinge loss')
plt.show()
```



In [105]:

```
C_opt = 10
c_opt = 1
d_opt = 4
```

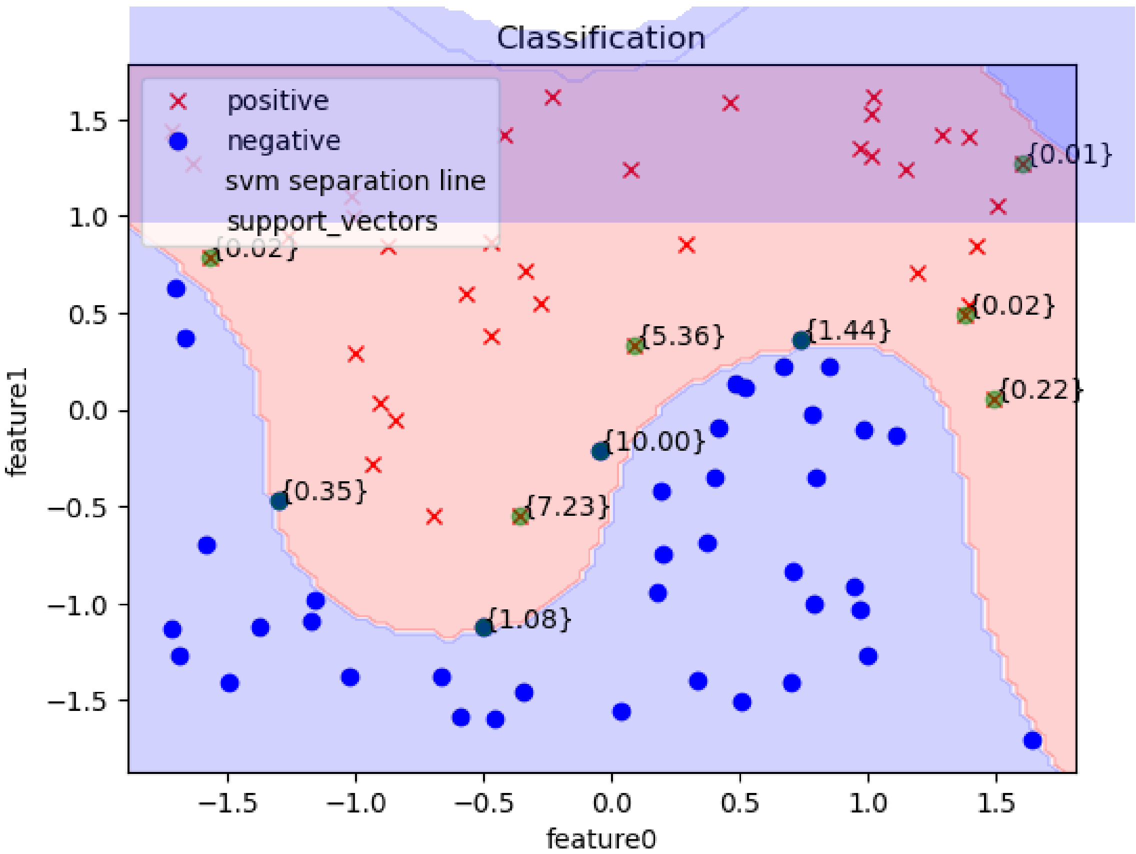
In [106]:
```
K_train =  polynomial_kernel(X_train_norm,X_train_norm,c_opt, d_opt)
alpha = SVM_dual(X_train_norm, y_train, C_opt, K_train)
alpha = np.array(alpha).reshape(X_train.shape[0],1)
```

```
     pcost       dcost       gap    pres   dres
 0: -7.7833e+01 -5.2860e+03  1e+04  7e-01  4e-13
 1:  9.5174e+00 -1.7254e+03  3e+03  1e-01  4e-13
 2:  2.9745e+01 -3.6486e+02  6e+02  2e-02  8e-14
 3:  5.1854e+00 -8.1707e+01  1e+02  3e-03  3e-14
 4: -7.6442e+00 -2.6008e+01  2e+01  5e-04  2e-14
 5: -1.2769e+01 -1.8905e+01  6e+00  7e-05  1e-14
 6: -1.4636e+01 -1.5472e+01  8e-01  1e-06  2e-14
 7: -1.4906e+01 -1.5102e+01  2e-01  2e-07  2e-14
 8: -1.4990e+01 -1.5006e+01  2e-02  3e-15  2e-14
 9: -1.4997e+01 -1.4997e+01  3e-04  3e-15  1e-14
10: -1.4997e+01 -1.4997e+01  3e-06  3e-15  2e-14
Optimal solution found.
```

In [107]:
```
support_id = (np.logical_and((alpha > 1e-5), (alpha<C_opt)).T)[0];
support_y = y_train[support_id][0]
b = (1/support_y - np.sum(alpha*y_train*((K_train[:,support_id])[:,0])))
```

In [121]:
```
plt.figure()
label_1 = ((y_train == 1).T)[0]
label_minus_1 = ((y_train == -1).T)[0]
plt.plot(X_train_norm[label_1,0],X_train_norm[label_1,1], 'rx')
plt.plot(X_train_norm[label_minus_1,0],X_train_norm[label_minus_1,1], 'bo')
def plot_decision_boundary( xmin, xmax, ymin, ymax):
  xx, yy = np.meshgrid(
      np.linspace(xmin, xmax, num=100, endpoint=True),
      np.linspace(ymin, ymax, num=100, endpoint=True))
  K = polynomial_kernel(X_train_norm, np.c_[xx.ravel(), yy.ravel()],c_opt,d_
  Z = predict(K,alpha,y_train,b)
  Z = Z.reshape(xx.shape)
  cs = plt.contourf(xx, yy, Z, alpha=0.2, cmap='bwr')
xmin, xmax, ymin, ymax = plt.axis()
plot_decision_boundary( xmin, xmax, ymin, ymax)

for i, txt in enumerate(alpha):
    if(txt > 1e-5):
       plt.annotate('{%.2f}'%(txt), (X_train_norm[i,0], X_train_norm[i,1]))
       plt.plot(X_train_norm[i,0], X_train_norm[i,1],'go', alpha = 0.5)

plt.legend(['positive', 'negative','svm separation line','support_vectors']
plt.xlabel('feature0')
plt.ylabel('feature1');
plt.title('Classification')
```

Out[121]:
```
Text(0.5, 1.0, 'Classification')
```



Vidimo da je granica malo pomerena na ustrb plavih primera iako je mogla 100% da klasifikuje sve primere da ne bi doslo do preobucavanja uslad loseg klasifikovanja crvenih primera na test skupu.

In [111]:
```
K_test = polynomial_kernel(X_train_norm,X_test_norm,c_opt,d_opt)
y_pred_train = predict(K_train, alpha, y_train,b).T
acc_train = np.sum(y_pred_train == y_train)/len(y_train)
y_test = y_test.reshape(X_test.shape[0],1)
y_pred_test = predict(K_test, alpha, y_train,b).T
acc_test = np.sum(y_pred_test == y_test)/len(y_test)
train_loss = hinge_loss(y_pred_train, y_train)
test_loss = hinge_loss(y_pred_test, y_test)
print(acc_train)
print(acc_test)
print(train_loss)
print(test_loss)
```

```
0.95
0.9
```

Krajnja tacnost polinomijalnog modela je 95% na obucavajucem skupu i 90% na test skupu.

Mozemo videti da je i gubitak manji nego kod linearnog kernela i da je malo veci na testirajucem skupu.

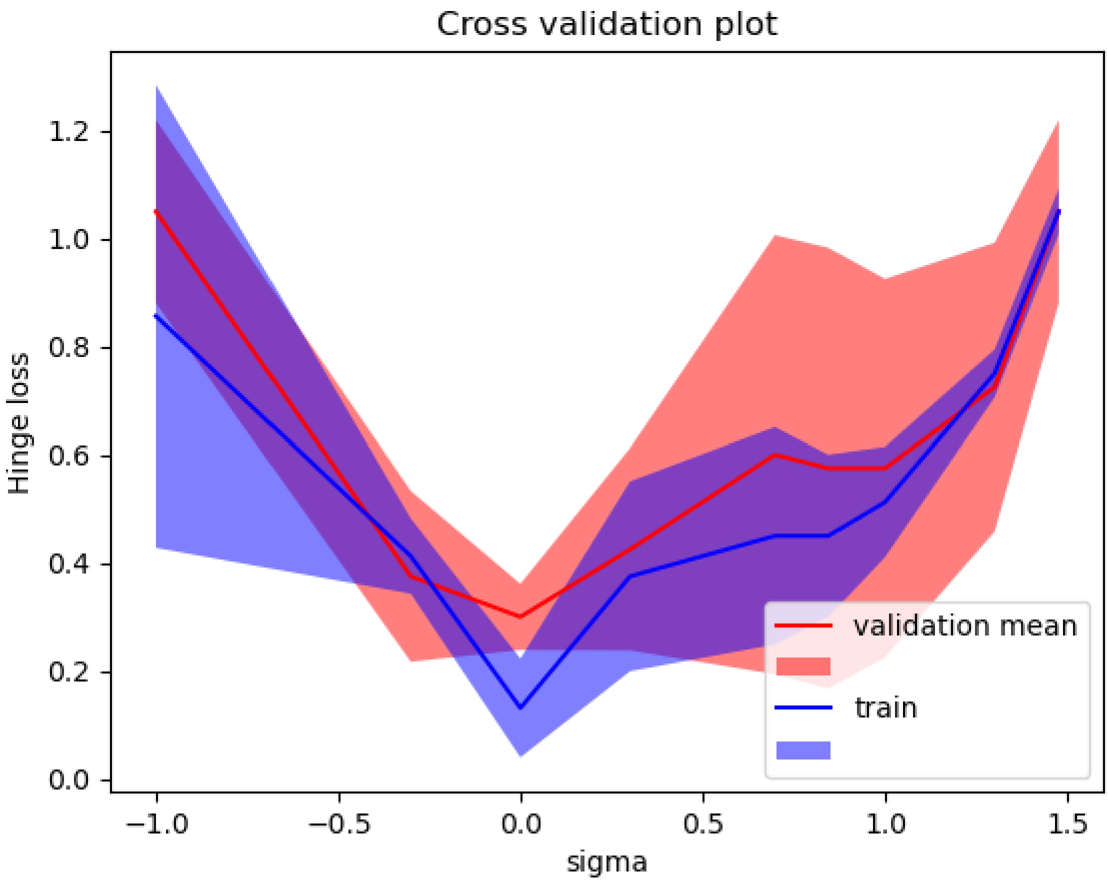
Odredjivanje hiperparametara gausovog kernela

In [115]:
```
all_params = {'C':10, 'c' : 1, 'd' : 4, 'sigma' : 1 }
var_params = [0.1,0.5,1,2,5,7,10, 20,30]
var_param_name = 'sigma'
kernel_type = 'G'
```

In [116]:
```
(validation_rmse_mean,validation_rmse_std,train_rmse_mean,train_rmse_std) =
```

```
     pcost       dcost       gap    pres   dres
 0:  2.1940e+02 -1.5652e+03  2e+03  2e-15  2e-15
 1:  2.0366e+01 -1.7714e+02  2e+02  2e-15  1e-15
 2: -2.5680e+01 -4.9069e+01  2e+01  8e-15  5e-16
 3: -2.8161e+01 -2.9322e+01  1e+00  3e-15  2e-16
 4: -2.8178e+01 -2.8241e+01  6e-02  1e-15  1e-16
 5: -2.8178e+01 -2.8179e+01  1e-03  4e-15  1e-16
 6: -2.8178e+01 -2.8178e+01  1e-05  3e-15  1e-16
Optimal solution found.
     pcost       dcost       gap    pres   dres
 0:  2.2743e+02 -1.6854e+03  2e+03  1e-15  2e-15
 1:  2.4535e+01 -1.8309e+02  2e+02  7e-15  1e-15
 2: -2.4511e+01 -4.9219e+01  2e+01  4e-15  4e-16
 3: -2.7304e+01 -2.8628e+01  1e+00  3e-16  2e-16
 4: -2.7339e+01 -2.7446e+01  1e-01  2e-16  1e-16
 5: -2.7344e+01 -2.7348e+01  3e-03  7e-16  1e-16
 6: -2.7345e+01 -2.7345e+01  1e-04  4e-16  1e-16
 7: -2.7345e+01 -2.7345e+01  2e-06  1e-15  9e-17
Optimal solution found.
```

In [118]:
```
plt.figure()
plt.plot(np.log10(var_params),validation_rmse_mean,c='r')
plt.fill_between(np.log10(var_params),validation_rmse_mean-validation_rmse_s
plt.plot(np.log10(var_params),train_rmse_mean, c= 'b')
plt.fill_between(np.log10(var_params),train_rmse_mean-train_rmse_std,train_r
plt.legend(['validation mean', '','train',''])
plt.title("Cross validation plot")
plt.xlabel('sigma')
plt.ylabel('Hinge loss')
plt.show()
```

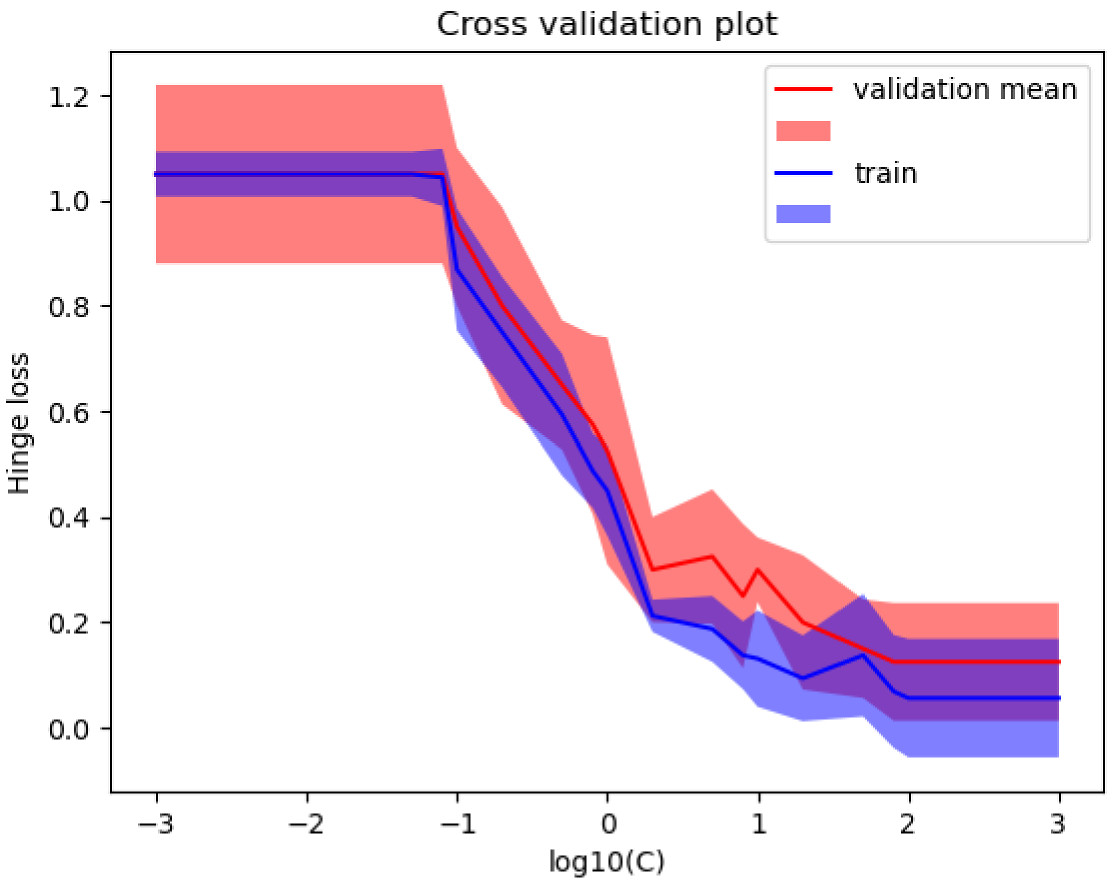


In [143]:
```
all_params = {'C':20, 'c' : 2, 'd' : 3, 'sigma' : 1 }
var_params =  [0.001,0.002,0.005,0.008,0.01,0.02,0.05,0.08,0.1,0.2,0.5,0.8,1
var_param_name = 'C'
kernel_type = 'G'
```

In [144]:
```
(validation_rmse_mean,validation_rmse_std,train_rmse_mean,train_rmse_std) =
```

```
     pcost       dcost       gap    pres   dres
 0: -9.7652e+00 -2.2225e+00  4e+02  2e+01  4e-16
 1: -2.5725e+00 -8.9610e-01  2e+01  1e+00  7e-16
 2: -1.1729e-01 -1.5026e-01  7e-01  3e-02  1e-15
 3: -5.0760e-02 -1.2272e-01  7e-02  4e-18  7e-16
 4: -5.6253e-02 -6.8871e-02  1e-02  1e-18  7e-16
 5: -5.7777e-02 -5.8162e-02  4e-04  2e-18  6e-16
 6: -5.7808e-02 -5.7890e-02  8e-05  2e-18  5e-16
 7: -5.7826e-02 -5.7849e-02  2e-05  2e-18  5e-16
 8: -5.7832e-02 -5.7835e-02  3e-06  2e-18  5e-16
 9: -5.7833e-02 -5.7834e-02  8e-07  1e-18  5e-16
10: -5.7833e-02 -5.7833e-02  6e-08  2e-18  6e-16
Optimal solution found.
     pcost       dcost       gap    pres   dres
 0: -1.0126e+01 -2.3395e+00  3e+02  2e+01  5e-16
 1: -2.5895e+00 -8.4808e-01  2e+01  1e+00  5e-16
 2: -1.0109e-01 -1.4627e-01  6e-01  3e-02  2e-15
 3: -5.1301e-02 -1.1529e-01  6e-02  4e-18  8e-16
 4: -5.8164e-02 -6.7831e-02  1e-02  1e-18  6e-16
```

In [145]:
```
plt.figure()
plt.plot(np.log10(var_params),validation_rmse_mean,c='r')
plt.fill_between(np.log10(var_params),validation_rmse_mean-validation_rmse_s
plt.plot(np.log10(var_params),train_rmse_mean, c= 'b')
plt.fill_between(np.log10(var_params),train_rmse_mean-train_rmse_std,train_r
plt.legend(['validation mean', '','train',''])
plt.title("Cross validation plot")
plt.xlabel('log10(C)')
plt.ylabel('Hinge loss')
plt.show()
```

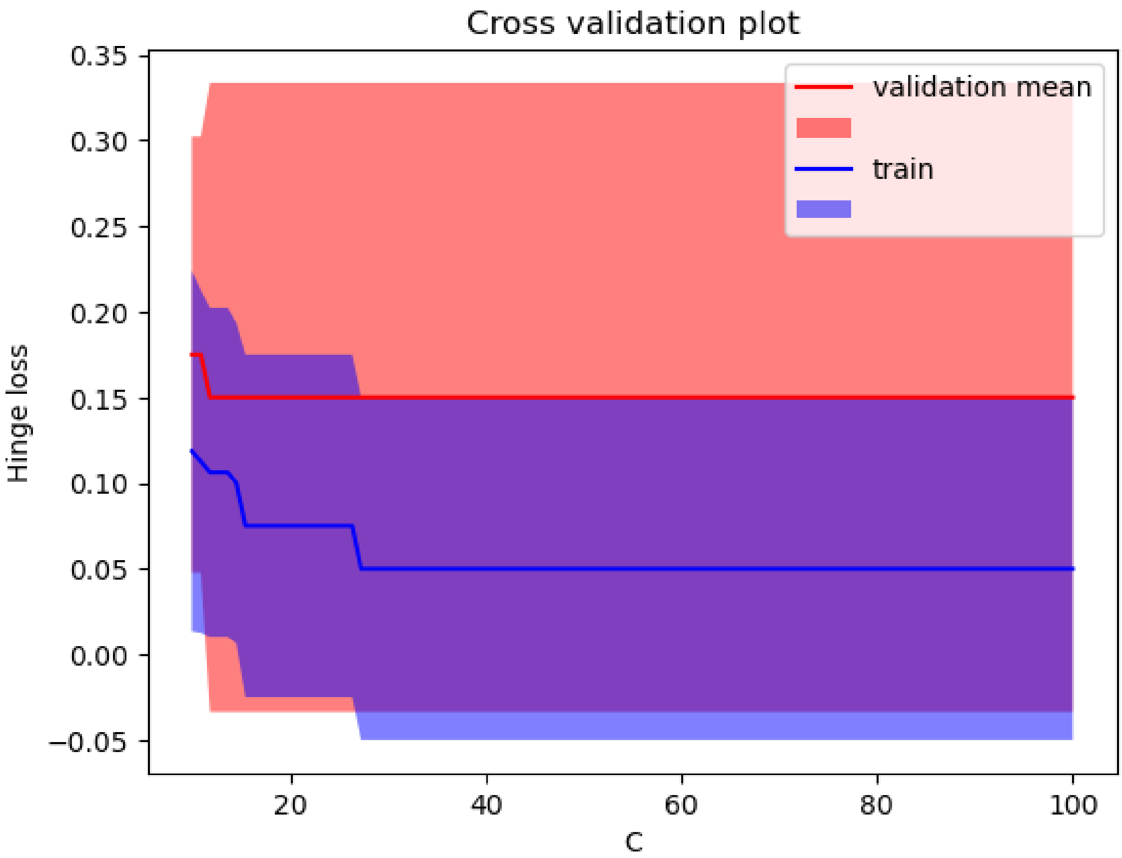


In [146]:
```
all_params = {'C':20, 'c' : 1, 'd' : 3, 'sigma' : 1 }
var_params =  list(np.linspace(10,100,100))
var_param_name = 'C'
kernel_type = 'P'
```

```
In [147]: (validation_rmse_mean,validation_rmse_std,train_rmse_mean,train_rmse_std) =
```

```
     pcost       dcost       gap    pres   dres
 0: -5.2775e+01 -3.7813e+03  9e+03  6e-01  9e-14
 1:  3.9208e+01 -1.1037e+03  2e+03  9e-02  9e-14
 2:  2.3736e+01 -2.2129e+02  3e+02  1e-02  4e-14
 3: -6.2432e+00 -5.5159e+01  5e+01  1e-03  2e-14
 4: -1.6515e+01 -3.0650e+01  2e+01  3e-04  1e-14
 5: -2.1269e+01 -2.3119e+01  2e+00  1e-06  1e-14
 6: -2.1947e+01 -2.2558e+01  6e-01  3e-07  9e-15
 7: -2.2222e+01 -2.2264e+01  4e-02  1e-08  1e-14
 8: -2.2241e+01 -2.2241e+01  5e-04  1e-10  9e-15
 9: -2.2241e+01 -2.2241e+01  5e-06  1e-12  1e-14
Optimal solution found.
     pcost       dcost       gap    pres   dres
 0: -4.3931e+01 -4.6796e+03  1e+04  8e-01  6e-14
 1:  5.4511e+01 -1.7909e+03  3e+03  1e-01  6e-14
 2:  3.9033e+01 -2.4306e+02  4e+02  1e-02  2e-14
 3: -1.4850e+00 -6.3431e+01  7e+01  2e-03  1e-14
 4: -1.5166e+01 -3.1320e+01  2e+01  4e-04  8e-15
 5: -2.0813e+01 -2.3533e+01  3e+00  2e-06  8e-15
```

In [148]:
```
plt.figure()
plt.plot(var_params,validation_rmse_mean,c='r')
plt.fill_between(var_params,validation_rmse_mean-validation_rmse_std,valida
plt.plot(var_params,train_rmse_mean, c= 'b')
plt.fill_between(var_params,train_rmse_mean-train_rmse_std,train_rmse_mean+
plt.legend(['validation mean', '','train',''])
plt.title("Cross validation plot")
plt.xlabel('C')
plt.ylabel('Hinge loss')
plt.show()
```



In [149]:
```
C_opt = 10
```

In [150]:
```
sigma_opt = 1;
```

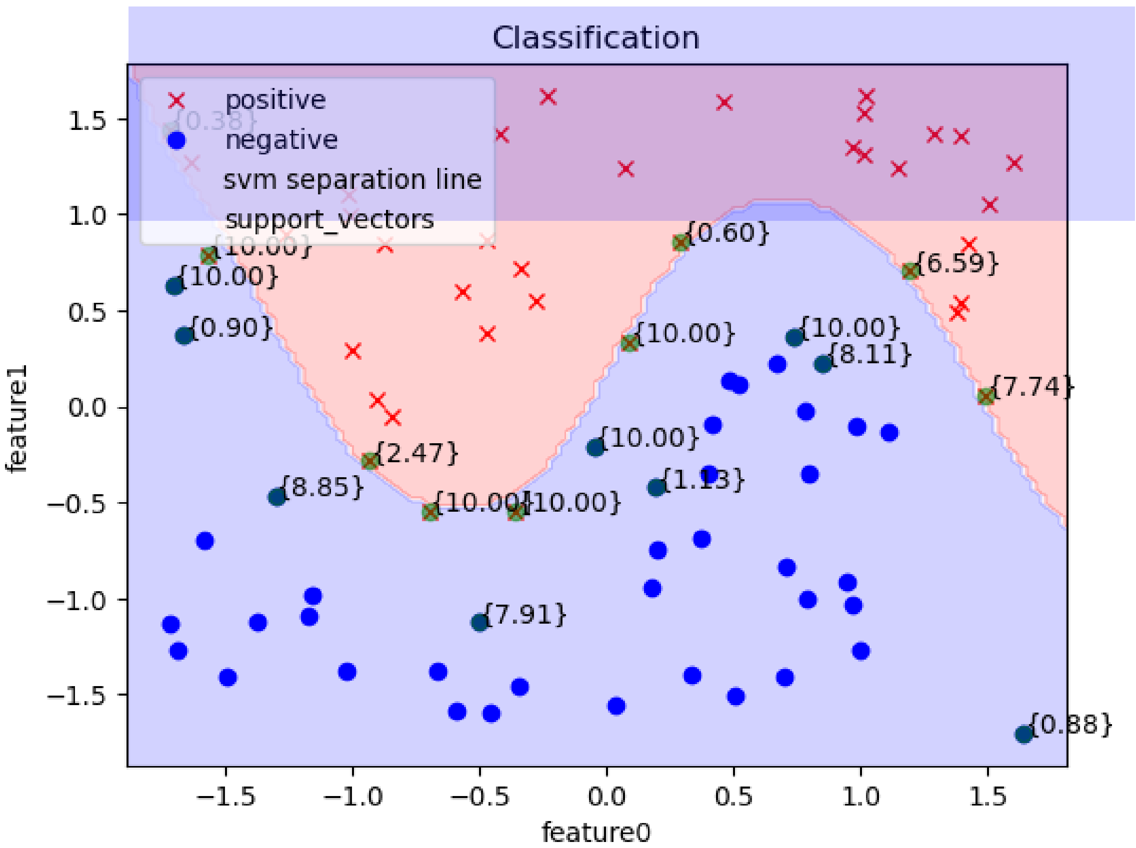
In [151]:
```
K_train =  gaussian_kernel(X_train_norm,X_train_norm,sigma_opt)
alpha = SVM_dual(X_train_norm, y_train, C_opt, K_train)
alpha = np.array(alpha).reshape(X_train.shape[0],1)
```

```
     pcost       dcost       gap    pres   dres
 0:  1.4765e+02 -2.8932e+03  4e+03  2e-01  4e-15
 1:  4.9726e+01 -3.7718e+02  5e+02  1e-02  3e-15
 2: -2.7016e+01 -1.5880e+02  1e+02  3e-03  2e-15
 3: -5.3669e+01 -1.0959e+02  6e+01  7e-04  3e-15
 4: -6.5864e+01 -9.3607e+01  3e+01  2e-04  2e-15
 5: -7.1632e+01 -8.2138e+01  1e+01  5e-05  3e-15
 6: -7.4732e+01 -7.6493e+01  2e+00  3e-06  3e-15
 7: -7.5339e+01 -7.5520e+01  2e-01  2e-07  3e-15
 8: -7.5410e+01 -7.5412e+01  2e-03  3e-09  3e-15
 9: -7.5411e+01 -7.5411e+01  2e-05  3e-11  3e-15
Optimal solution found.
```

In [152]:
```
support_id = (np.logical_and((alpha > 1e-5), (alpha<C_opt)).T)[0];

support_y = y_train[support_id][0]
b = (1/support_y - np.sum(alpha*y_train*((K_train[:,support_id])[:,0])))
```

In [153]:
```
plt.figure()
label_1 = ((y_train == 1).T)[0]
label_minus_1 = ((y_train == -1).T)[0]
plt.plot(X_train_norm[label_1,0],X_train_norm[label_1,1], 'rx')
plt.plot(X_train_norm[label_minus_1,0],X_train_norm[label_minus_1,1], 'bo')
def plot_decision_boundary( xmin, xmax, ymin, ymax):
  xx, yy = np.meshgrid(
      np.linspace(xmin, xmax, num=100, endpoint=True),
      np.linspace(ymin, ymax, num=100, endpoint=True))
  K = gaussian_kernel(X_train_norm, np.c_[xx.ravel(), yy.ravel()],sigma_opt
  Z = predict(K,alpha,y_train,b)
  Z = Z.reshape(xx.shape)
  cs = plt.contourf(xx, yy, Z, alpha=0.2, cmap='bwr')
xmin, xmax, ymin, ymax = plt.axis()
plot_decision_boundary( xmin, xmax, ymin, ymax)

for i, txt in enumerate(alpha):
    if(txt > 1e-5):
       plt.annotate('{%.2f}'%(txt), (X_train_norm[i,0], X_train_norm[i,1]))
       plt.plot(X_train_norm[i,0], X_train_norm[i,1],'go', alpha = 0.5)

plt.legend(['positive', 'negative','svm separation line','support_vectors']
plt.xlabel('feature0')
plt.ylabel('feature1');
plt.title('Classification')
```

Out[153]: `Text(0.5, 1.0, 'Classification')`



In [154]:
```
K_test = gaussian_kernel(X_train_norm,X_test_norm,sigma_opt)
y_pred_train = predict(K_train, alpha, y_train,b).T
acc_train = np.sum(y_pred_train == y_train)/len(y_train)
y_test = y_test.reshape(X_test.shape[0],1)
y_pred_test = predict(K_test, alpha, y_train,b).T
acc_test = np.sum(y_pred_test == y_test)/len(y_test)
train_loss = hinge_loss(y_pred_train, y_train)
test_loss = hinge_loss(y_pred_test, y_test)
print(acc_train)
print(acc_test)
print(train_loss)
print(test_loss)
```

```
0.95
0.85
```

Krajnja tacnost gausovog kernela je 95% na obucavajucem skupu i 85% na test skupu

Uporedni prikaz svih modela

In [158]:
```
import pandas as pd
df = {"train":[0.9, 0.95, 0.95], "test":[0.85, 0.90, 0.85]}
df = pd.DataFrame.from_dict(df)
df.index = ["linear", "polynomial","gaussian"]
df
```

Out[158]:

	train	test
linear	0.90	0.85
polynomial	0.95	0.90
gaussian	0.95	0.85

Vidimo da se kao najbolji pokazao polinomijalni kernel, jer daje najbolje tacnosti i najmanje se preobucio.