SVM and Logitic regression implementation:

Importing panda and numpy:

In [47]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(20)
```

Importing the dataset and converting as panda framework:

In [48]:
```
def import_data (filename):
    """
    This function, imports the train/test data and create the attribute matrix ar
    """
    Matrix = []
    Label = []
    with open(filename) as f:

        for line in f:
            sample = line.split()
            Label.append(float(sample[0]))
            sample.pop(0)
            row = []
            for s in sample:
                feature, value = s.split(':')
                z = len(row)
                nz = int(feature) - (z+1)
                for i in range (nz):
                    row.append(0)
                row.append(float(value))
            Matrix.append(row)
    data =[]
    M = max(len(row) for row in Matrix)
    #print("M:",M)
    for row in Matrix:
        nz = M - (len(row))
        for i in range (nz):
            row.append(0)
        data.append(row)
    Label1 = np.array(Label)
    data1= np.array(data)
    #print("aaa:",Label1, data1.shape)
    S1 = np.concatenate((data1, Label1[:,None]),axis=1)
    attributes = np.arange(1, np.size(data1,1)+2)
    #print(attributes)
    samples = range(0,np.size(data1,0))
    data2 = pd.DataFrame(S1, columns=attributes, index=samples)
    #print('label',data2[6])

    return data2
    #print("data1:",data1.shape)
```

Fold database creation:

In [49]:
```
def update_label(D):
    x,y = D.shape
    for i in range(x):
        if D[y][i] ==0.0:
            D[y][i] = -1.0
    return (D)
def k_fold(D,k):
    cols = D.columns
    D = D.to_numpy()
    r_n, _ = D.shape
    k_n = (r_n//5)
    lb = (k-1)*k_n
    if k == 5:
        ub = r_n
    else:
        ub = k*k_n-1

    fk = D [lb:ub, :]

    Fk = pd.DataFrame(fk, columns=cols)
    return Fk

def import_label (D, new_feature):
    D = D.to_numpy()
    D = D.copy()
    new_feature = new_feature.to_numpy()
    labels = D[:, -1]
    labels = labels[:,None]

    D_out = np.append(new_feature, labels, axis=1)

    attributes = np.arange(1, np.size(D_out,1)+1)
    D_out = pd.DataFrame(D_out, columns=attributes)

    return D_out
def concat_datasets (D1, D2):
    if type(D1) != np.ndarray:
        D1 = D1.to_numpy()
    if type(D2) != np.ndarray:
        D2 = D2.to_numpy()
    D1 = D1.copy()
    D_out = np.append(D1[:,:-1], D2, axis=1)

    attributes = np.arange(1, np.size(D_out,1)+1)
    D_out = pd.DataFrame(D_out, columns=attributes)
    return D_out
```

Importing the tfidf datasets:

In [50]:
```
Train_data1 = import_data('tfidf.train.libsvm')
Train_data_tfidf = update_label(Train_data1)
Test_data1 = import_data('tfidf.test.libsvm')
Test_data_tfidf = update_label(Test_data1)
Eval_data_tfidf = import_data('tfidf.eval.anon.libsvm')
```

Importing the miscellaneous datasets:

In [51]:
```
misc_train = pd.read_csv ('misc-attributes-train.csv')
train_samples, _ = misc_train.shape
misc_test = pd.read_csv ('misc-attributes-test.csv')
test_samples, _ = misc_test.shape
misc_eval = pd.read_csv ('misc-attributes-eval.csv')
eval_samples, _ = misc_eval.shape
```

In order to convert the database to one hot encoding, all the dataset are concatenated and converted to correlate the cominations.

In [52]:
```
database = pd.concat([misc_train, misc_test, misc_eval], axis=0)
```

In [9]:
```
database
```

Out[9]:

	defendant_age	defendant_gender	num_victims	victim_genders	offence_category	offence_su
0	62	female	1	male	theft	theft
1	17	male	1	male	theft	po
2	not known	male	1	male	theft	po
3	not known	male	1	male	theft	sim
4	52	male	1	female	theft	po
...	...	...	...	...	...	
5245	not known	male	1	male	theft	theft
5246	not known	male	0	NaN	sexual	
5247	not known	male	1	male	theft	stealingF
5248	26	male	1	male	theft	
5249	16	male	1	female	theft	sim

25000 rows × 6 columns

In [10]:
```
database.dtypes
```

Out[10]:
```
defendant_age           object
defendant_gender        object
num_victims              int64
victim_genders          object
offence_category        object
offence_subcategory     object
dtype: object
```

In [53]:
```
database[database.isnull().any(axis=1)]
# Converting "NaN" to no_gender in victom_genders category:
database = database.fillna({"victim_genders": "no_gender"})
database.head()

# convert all string data in defendant such as not known ,... to Nan and then sul
database['defendant_age'] = pd.to_numeric(database.defendant_age, errors='coerce
database = database.fillna({"defendant_age": 0})
database
```

Out[53]:

	defendant_age	defendant_gender	num_victims	victim_g
0	62.0	female	1	
1	17.0	male	1	
2	0.0	male	1	
3	0.0	male	1	
4	52.0	male	1	
5	40.0	male	0	no_
6	0.0	male	1	
7	0.0	female	1	
8	30.0	male	0	no_
9	23.0	male	1	
10	30.0	male	1	

In [54]:
```
# Now that all the data are free of Nan we can convert them to one-hot encoding.
misc_transfered = pd.concat([database.defendant_age, database.num_victims, pd.ge
# for dicision tree i convert all of the featres to one-hot encoding
misc_transfered_all_bin = pd.concat([pd.get_dummies(database.defendant_age), pd.
# Label encoding:
database["defendant_gender"] = database["defendant_gender"].astype('category')
database["victim_genders"] = database["victim_genders"].astype('category')
database["offence_category"] = database["offence_category"].astype('category')
database["offence_subcategory"] = database["offence_subcategory"].astype('catego

misc_transfered_le = pd.concat([database.defendant_age, database.num_victims, da
```

In [49]:
```
misc_transfered
```

Out[49]:

	defendant_age	num_victims	female	indeterminate	male	female	female;female	female;
0	62.0	1	1	0	0	0	0	
1	17.0	1	0	0	1	0	0	
2	0.0	1	0	0	1	0	0	
3	0.0	1	0	0	1	0	0	
4	52.0	1	0	0	1	1	0	
5	40.0	0	0	0	1	0	0	
6	0.0	1	0	0	1	1	0	
7	0.0	1	1	0	0	0	0	
8	30.0	0	0	0	1	0	0	
9	23.0	1	0	0	1	0	0	
10	30.0	1	0	0	1	0	0	

In [62]:
```
misc_transfered_le.head()
```

Out[62]:

	defendant_age	num_victims	0	1	2	3
0	62.0	1	0	33	7	49
1	17.0	1	2	33	7	34
2	0.0	1	2	33	7	34
3	0.0	1	2	33	7	45
4	52.0	1	2	0	7	34

In [55]:
```
Train_misc_transfered = misc_transfered.iloc[:train_samples,:]
Test_misc_transfered = misc_transfered.iloc[train_samples:train_samples+test_sam
Eval_misc_transfered = misc_transfered.iloc[train_samples+test_samples:,:]
```

In [56]:
```
Train_misc = import_label(Train_data_tfidf, Train_misc_transfered)
Test_misc = import_label(Test_data_tfidf, Test_misc_transfered)
Eval_misc = import_label(Eval_data_tfidf, Eval_misc_transfered)
print(Train_misc.shape)
```

```
(17500, 140)
```

Cross validation procedure:

In [57]:
```
def cross_val(f1, f2, f3, f4, f5, max_epoch, learning_rate, C = 0, learning_stra
    """
    The function calculates the mean accuracy and std based on the 5-fold cross v
    """

    #train_data = pd.DataFrame(columns = f1.columns)
    dataset = []
    acc = []
    loss = []

    for i in range (1,6):
        valid_data = eval("f"+str(i))
        train_name =[]
        val_name = ["f"+str(i)]
        #print(i,val_name)
        #print(valid_data)
        for j in range(1,6):
            if j != i:
                #print(j)
                train_name.append ("f"+str(j))
                dataset.append(eval("f"+str(j)))
        train_data = pd.concat(dataset, ignore_index=True)
        dataset = []
        #print(train_data)
        if learning_strategy == 'SVM':
            w, b, _ = SVM (train_data, max_epoch, learning_rate, C)
        elif learning_strategy == 'Logestic regression':
            w, b, _ = log_reg (train_data, max_epoch, learning_rate, C)

        w = w[-1]
        #print(w)
        b = b [-1]

        #print(train_name)
        ac, lo = accuracy (valid_data, w, b, learning_strategy, C, C)
        acc.append (ac)
        loss.append (lo)
    #print("accuracy:", acc)
    Mean_acc = np.mean(acc)
    Mean_loss = np.mean(loss)

    return Mean_acc, Mean_loss
```

In [58]:
```
def accuracy (D, w, b, loss_metric = "SVM", C = 1, sig2=0):
    if loss_metric == "Logestic regression":
        C = 1

    #print(sig2, C)

    """
        This function returns the accuracy of the dataset based on set D and wei
    """
    if type(D) != np.ndarray:
        D = D.to_numpy()
    n_correct_prediction = 0
    n_samples = np.size(D,0)
    label_ix = np.size(D,1)
    if loss_metric == "SVM":
        loss = .5 *(np.dot(w,w) + b*b)
    elif loss_metric == "Logestic regression":
        loss = (np.dot(w,w) + b*b)/sig2
        #print("loss",loss)
    for i in range(n_samples):
        sample = D[i,:]
        true_label = sample[-1]
        xi = sample[:-1]
        dot_product = np.dot(xi,w) + b
        predicted_label = np.sign (dot_product)
        if loss_metric == "SVM":
            loss += np.max([0, 1.0 - true_label * dot_product])
        elif loss_metric == "Logestic regression":
            loss += np.log (1 + np.exp(-true_label * dot_product))
        if predicted_label == true_label:
            n_correct_prediction += 1
    acc = n_correct_prediction/n_samples * 100
    loss = C*loss
    return acc, loss

def prediction (D, w, b):
    """
        This function returns the prediction of the dataset based on set D and we
    """
    D = D.to_numpy()
    n_samples = np.size(D,0)
    label_ix = np.size(D,1)
    pred = []
    for i in range(n_samples):
        sample = D[i,:]
        xi = sample[:-1]
        predicted_label = np.sign (np.dot(xi,w) + b)
        #print(predicted_label[0])
        if predicted_label == -1.0:
            predicted_label = [0.0]
        pred.append([i, predicted_label[0]])

    Pred = pd.DataFrame(pred, columns=['example_id', 'label'])

    return Pred
```

SVM Implementation:

In [13]:

```
def SVM (D, max_epoch, learning_rate, Cost, threshold=0.0005):
    c = Cost
    if type(D) != np.ndarray:
        D = D.to_numpy()
    lr0 = learning_rate
    #print('training size:')

    w_size = np.size(D,1)-1
    w = -.01 + 0.02 * np.random.rand(w_size)
    #w = -.01 * np.ones(w_size)

    b = -.01 + 0.02 * np.random.rand(1)
    #b = -0.01
    update = 0
    ep_w = []
    ep_b = []
    ep_update = []
    train_loss = []
    losses = []
    j = 0
    for epoch in range(1, max_epoch+1):
        #1.shuffle the data
        lr = lr0/(1+epoch)
        np.random.shuffle(D)
        #2.Update weights:
        for i in range (np.size(D,0)):
            xi = D[i,:-1]
            yi = D[i,-1]
            if yi * (np.dot(xi, w) + b) <= 1:
                update += 1
                w = (1-lr)*w + lr * c * yi * xi
                b = (1-lr)*b + lr * c * yi
            else:
                w = (1-lr)*w
                b = (1-lr)*b

        #print("w0:", w[0])
        w1 = w
        b1 = b
        update1 = update
        _, loss = accuracy (D, w,b, "SVM", c)
        loss = loss[0]
        if epoch == 1:
            loss1 = loss
        train_loss.append(loss/loss1)
        losses.append(loss)

        # Stopping criteria:
        if epoch> 5:
            #print(epoch)
            #print(train_loss)
            if (abs(train_loss[epoch-1] - train_loss[epoch-2])) < threshold and
                j = 1
                #print(j)
                #print(i)
```

```
                break
        #print(b)
        #print(b1)
        ep_w.append(w1.copy())
        ep_b.append(b1.copy())
        ep_update.append(update1)
        #print('ep_b:',ep_b)
    #print('update:', update)
    ep_w = np.array(ep_w)
    ep_b = np.array(ep_b)
    ep_update = np.array(ep_update)
    return ep_w, ep_b, losses
```

SVM for miscalleneous data

In [59]:

```
# generating 5-fold dataset:
Data1 = Train_misc
cols = Data1.columns
Data1 = Data1.to_numpy()
np.random.shuffle(Data1)
Data1 = pd.DataFrame(Data1, columns=cols)
f1 = k_fold(Data1,1)
f2 = k_fold(Data1,2)
f3 = k_fold(Data1,3)
f4 = k_fold(Data1,4)
f5 = k_fold(Data1,5)
```

In [16]:
```
# Evaluating the network accuracy based on different values for learning rates an
"""
"""
Learning_rates = [10**(-2), 10**(-3), 10**(-4)]
cost = [10**3, 10**(2), 10**(1), 10**(0)]
max_epoch = 10
acc_mean = []
loss_mean = []
result = []


for lr in Learning_rates:
    for c in cost:
        mean_ac, mean_loss = cross_val(f1, f2, f3, f4, f5, max_epoch, lr, c, lea
        #print(mean_ac)
        acc_mean.append(mean_ac)
        loss_mean.append(mean_loss)
        result.append([lr, c, mean_ac, mean_loss])
        #print(lr, u)

result = np.array(result)
Best_lr = result[np.argmax(result[:,2]), 0]
Best_c = result[np.argmax(result[:,2]), 1]
best_acc = result[np.argmax(result[:,2]), 2]
best_loss = result [np.argmax(result[:,2]), 3]

print('Cross validation results for different Learning rates and loss tradeoff:'
result = pd.DataFrame(result, columns=['Learning rate', 'Loss tradeoff', 'accura

pd.set_option('display.max_rows', None)
print(result.to_string(index = False))
print('Best learning rate:', Best_lr)

print('Best Cost:', Best_c)

report1 = [{'Best learning rate':Best_lr, 'Best Loss tradeoff':Best_c, 'Best acc
report1 = pd.DataFrame.from_records(report1)
print(report1.to_string(index = False))
```

```
Cross validation results for different Learning rates and loss tradeoff:
 Learning rate  Loss tradeoff  accuracy mean     loss mean
        0.0100         1000.0      63.345057  3.198734e+08
        0.0100          100.0      64.614344  4.144688e+06
        0.0100           10.0      66.930462  4.902023e+04
        0.0100            1.0      74.256738  2.409852e+03
        0.0010         1000.0      63.705245  3.431039e+07
        0.0010          100.0      78.103551  4.014011e+05
        0.0010           10.0      78.057829  2.042510e+04
        0.0010            1.0      78.006382  2.271897e+03
        0.0001         1000.0      78.189288  3.026126e+06
        0.0001          100.0      77.972088  2.031631e+05
        0.0001           10.0      77.823481  1.998764e+04
        0.0001            1.0      78.023532  2.290888e+03
Best learning rate: 0.0001
Best Cost: 1000.0
```

```
   Best learning rate  Best Loss tradeoff  Best accuracy
               0.0001              1000.0      78.189288
```

In [17]:

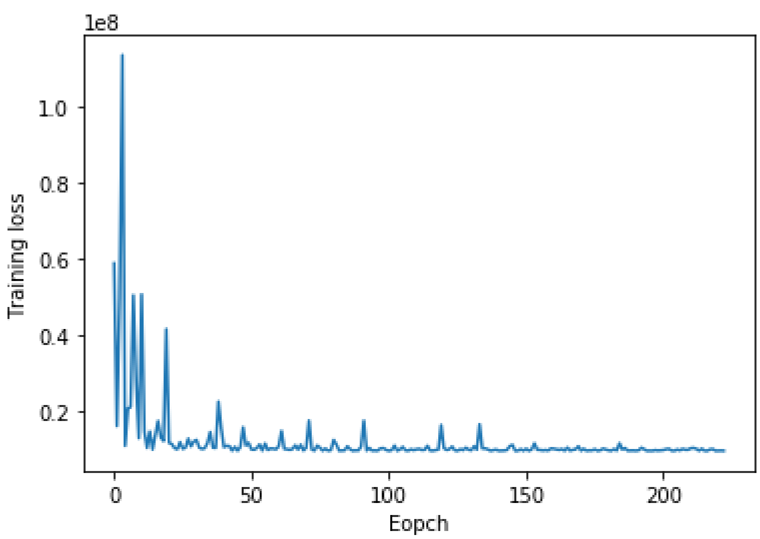
```
max_epoch = 500
threshold = 0.001
w1, b1, loss1 = SVM (Train_misc, max_epoch, Best_lr, Best_c, threshold)
train_acc = []
train_acc1 =[]
train_loss = []
acc = [0,0,0]
j = 0
print("max epoch:", len(b1))
for i in range (len(b1)):
    #print(i)
    #print(w[i][0])
    acc_, loss = accuracy (Train_misc, w1[i][:],b1[i], "SVM", Best_c)
    loss = loss[0]
    train_acc.append (acc_)
    acc[0] = i
    acc[1] = acc_
    acc[2] = loss
    train_acc1.append(acc.copy())
train_acc = np.array(train_acc)

best_epoch = np.argmax(train_acc) + 1

test_acc, test_loss =  accuracy (Test_misc, w1[best_epoch-1][:],b1[best_epoch-1]
```

```
max epoch: 223
```

In [18]:
```python
Epoch = np.arange(1,max_epoch+1)

data2 = pd.DataFrame(train_acc1, columns=['Epoch','Train accuracry', 'loss'])
print(data2.to_string(index = False))
train_acc1 = np.array(train_acc1)
```

```
 Epoch  Train accuracry          loss
     0        78.108571  5.864988e+07
     1        78.177143  1.592573e+07
     2        78.177143  4.959972e+07
     3        49.737143  1.134894e+08
     4        78.565714  1.072144e+07
     5        78.177143  2.056508e+07
     6        78.177143  2.096389e+07
     7        49.737143  5.040355e+07
     8        49.760000  2.775001e+07
     9        78.382857  1.273359e+07
    10        49.737143  5.061785e+07
    11        78.240000  1.458439e+07
    12        78.542857  1.018698e+07
    13        78.222857  1.453079e+07
    14        78.131429  9.791463e+06
    15        78.274286  1.357090e+07
    16        78.188571  1.729152e+07
    17        78.280000  1.297330e+07
```

In [19]:
```python
#b = plt.plot(train_acc1[:,0], train_acc1[:,3])
plt.xlabel('Eopch')
plt.ylabel('Training loss')
c = plt.plot(train_acc1[:,0], train_acc1[:,2])
```



In [20]:
```
report1 = [{'Best learning rate':Best_lr, 'Best loss tradeoff':Best_c, 'Best cro
            'Best epoch':best_epoch,
            'Train accuracy (%)':train_acc1[best_epoch-1][1],
            'Test accuracy (%)':test_acc}]

report1 = pd.DataFrame.from_records(report1)
print(report1.to_string(index = False))
```

```
 Best learning rate  Best loss tradeoff  Best cross val. acc. (%)  Best epoch
Train accuracy (%)  Test accuracy (%)
             0.0001              1000.0                 78.189288           5
78.565714          79.733333
```

In [21]:
```
pred1 = prediction (Eval_misc,  w1[best_epoch-1][:], b1[best_epoch-1])
#print(pred1)
pred1.to_csv ('svm_misc_labels.csv', index = False, header=True)
```

2. Logistic regression:

In [60]:

```
def log_reg (D, max_epoch, learning_rate, sigma2, threshold = 0.00001):
    if type(D) != np.ndarray:
        D = D.to_numpy()
    lr0 = learning_rate
    #print('training size:')
    z = np.max(D,0)
    #print(D.shape)
    #print(z)
    z = z[:-1]
    w_size = np.size(D,1)-1
    w = -.01 + 0.02 * np.random.rand(w_size)
    #w = -.01 * np.ones(w_size)
    #w = np.zeros(w_size)

    b = -.01 + 0.02 * np.random.rand(1)
    #b = 0
    #b = -0.01
    update = 0
    ep_w = []
    ep_b = []
    ep_update = []
    train_loss = []
    losses = []
    for epoch in range(1, max_epoch+1):
        #1.shuffle the data
        lr = lr0/(1+epoch)
        np.random.shuffle(D)
        #2.Update weights:
        for i in range (np.size(D,0)):
            xi = D[i,:-1]
            yi = D[i,-1]
            b1 = b
            z1 = yi*(np.dot(z, w) + b)
#             b1 = (1-2*lr/sigma2)*b + lr * yi /(1+np.exp(yi*(np.dot(xi, w) + b)
#             w = (1-2*lr/sigma2)*w + lr * yi * xi /(1+np.exp(yi*(np.dot(xi, w) 
#             b = b1
#             np.warnings.filterwarnings('ignore')
#             update += 1
#             print("w0:", w[0])
            #print(z1)
            if z1>-10.0 and z1<10.0:
                b1 = (1-2*lr/sigma2)*b + lr * yi /(1+np.exp(yi*(np.dot(xi, w) + b
                w = (1-2*lr/sigma2)*w + lr * yi * xi /(1+np.exp(yi*(np.dot(xi, w
                b = b1
                np.warnings.filterwarnings('ignore')
                update += 1
            elif z1<=-10.0:
                b = (1-2*lr/sigma2)*b + lr * yi
                w = (1-2*lr/sigma2)*w + lr * yi * xi
                np.warnings.filterwarnings('ignore')
            elif z1 >= 10.0:
                b = (1-2*lr/sigma2)*b
                w = (1-2*lr/sigma2)*w
                np.warnings.filterwarnings('ignore')
        w1 = w
```

```
        b1 = b
        update1 = update
        update1 = update
        _, loss = accuracy (D, w,b, "Logestic regression", 1, sigma2)
        loss = loss[0]
        if epoch == 1:
            loss1 = loss
        train_loss.append(loss/loss1)
        losses.append(loss)
        # Stopping criteria:
        if epoch> 5:
            #print(epoch)
            #print(train_loss)
            if (abs(train_loss[epoch-1] - train_loss[epoch-2])) < threshold and
                j = 1
                #print(j)
                #print(i)
                break
        #print(b)
        #print(b1)
        ep_w.append(w1.copy())
        ep_b.append(b1.copy())
        ep_update.append(update1)
        #print('ep_b:',ep_b)
    #print('update:', update)
    ep_w = np.array(ep_w)
    ep_b = np.array(ep_b)
    ep_update = np.array(ep_update)
    return ep_w, ep_b, losses
```

Logistic regression for miscellaneous

In [61]:
```
# generating 5-fold dataset:
Data1 = Train_misc
cols = Data1.columns
Data1 = Data1.to_numpy()
np.random.shuffle(Data1)
Data1 = pd.DataFrame(Data1, columns=cols)
f1 = k_fold(Data1,1)
f2 = k_fold(Data1,2)
f3 = k_fold(Data1,3)
f4 = k_fold(Data1,4)
f5 = k_fold(Data1,5)
```

In [62]:
```
# Evaluating the network accuracy based on different values for learning rates ar
"""
"""
Learning_rates = [10**(-2), 10**(-3), 10**(-4), 10**(-5)]
sigma2 = [10**1, 10**2, 10**3, 10**4]
max_epoch = 10
acc_mean = []
loss_mean = []
result = []
#Learning_rates = [1]
#sigma2 =  [0.1]
for lr in Learning_rates:
    for sig in sigma2:
        mean_ac, mean_loss = cross_val(f1, f2, f3, f4, f5, max_epoch, lr, sig, l
        #print(mean_ac)
        acc_mean.append(mean_ac)
        loss_mean.append(mean_loss)
        result.append([lr, sig, mean_ac, mean_loss])
        #print(lr, u)

result = np.array(result)
Best_lr = result[np.argmax(result[:,2]), 0]
Best_sigma2 = result[np.argmax(result[:,2]), 1]
best_acc = result[np.argmax(result[:,2]), 2]
best_loss = result [np.argmax(result[:,2]), 3]

print('Cross validation results for different Learning rates and loss tradeoff:'
result = pd.DataFrame(result, columns=['Learning rate', 'Sigma2', 'accuracy mean

pd.set_option('display.max_rows', None)
print(result.to_string(index = False))
print('Best learning rate:', Best_lr)

print('Best sigma2:', Best_sigma2)

report1 = [{'Best learning rate':Best_lr, 'Best sigma2':Best_sigma2, 'Best accura
report1 = pd.DataFrame.from_records(report1)
print(report1.to_string(index = False))
```

```
Cross validation results for different Learning rates and loss tradeoff:
 Learning rate   Sigma2  accuracy mean    loss mean
       0.01000     10.0      78.086371  2076.998204
       0.01000    100.0      77.760611  1959.519299
       0.01000   1000.0      76.903285  1920.253789
       0.01000  10000.0      75.234163  2033.469735
       0.00100     10.0      78.069223  2022.479549
       0.00100    100.0      78.097803  1959.349949
       0.00100   1000.0      77.834906  1938.556520
       0.00100  10000.0      77.589105  1942.311658
       0.00010     10.0      78.074939  2046.277839
       0.00010    100.0      78.040643  2033.954846
       0.00010   1000.0      78.057791  2032.154826
       0.00010  10000.0      78.040643  2031.460703
       0.00001     10.0      78.092087  2128.521090
       0.00001    100.0      78.092087  2128.033814
```

```
        0.00001   1000.0      78.092087  2129.855284
        0.00001  10000.0      78.092087  2129.211079
Best learning rate: 0.001
Best sigma2: 100.0
 Best learning rate  Best sigma2  Best accuracy
              0.001        100.0      78.097803
```

In [63]:

```
#Train_data = import_data('train')
max_epoch = 500
threshold = 0.0001
w, b, loss = log_reg (Train_misc, max_epoch, Best_lr, Best_sigma2, threshold)

#print(b)
train_acc = []
train_acc1 =[]
acc = [0,0,0]
j = 0
print("max epoch:", len(b))
for i in range (len(b)):
    #print(w[i][0])
    acc1, loss = accuracy (Train_misc, w[i][:],b[i], "Logestic regression", 1, B
    loss = loss[0]
    train_acc.append (acc1)
    acc[0] = i
    acc[1] = acc1
    acc[2] = loss
    train_acc1.append(acc.copy())
train_acc = np.array(train_acc)

best_epoch = np.argmax(train_acc) + 1

#Test_data = import_data('test')

test_acc, test_loss =  accuracy (Test_misc, w[best_epoch-1][:],b[best_epoch-1],
```

```
max epoch: 95
```

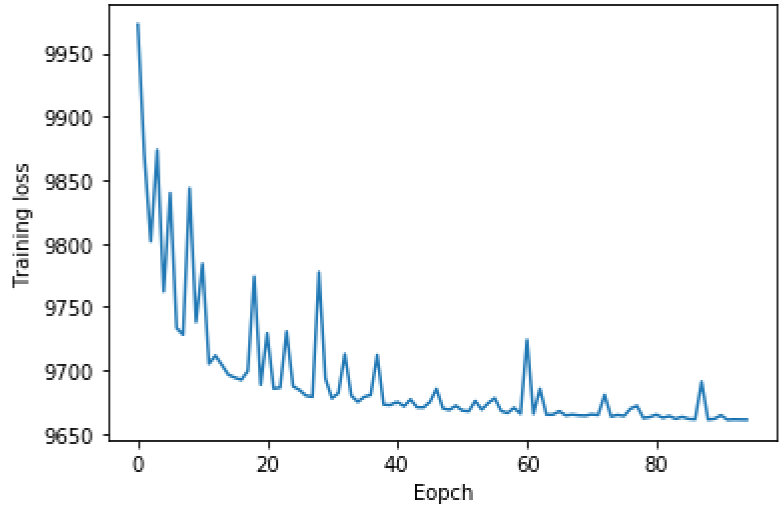
In [64]:
```
Epoch = np.arange(1,max_epoch+1)

data2 = pd.DataFrame(train_acc1, columns=['Epoch','Train accuracry', 'loss'])
print(data2.to_string(index = False))
train_acc1 = np.array(train_acc1)
```

```
Epoch  Train accuracry         loss
    0        77.954286  9972.529946
    1        78.005714  9869.826979
    2        78.034286  9801.618649
    3        78.091429  9873.675303
    4        78.068571  9762.069347
    5        77.537143  9839.729869
    6        78.051429  9733.189267
    7        77.925714  9728.005167
    8        77.308571  9843.509593
    9        78.080000  9737.996284
   10        78.091429  9783.989392
   11        77.982857  9705.081392
   12        77.868571  9711.541337
   13        77.885714  9704.172378
   14        77.982857  9696.482176
   15        77.971429  9694.192693
   16        77.965714  9692.397431
   17        78.045714  9699.341020
   18        77.531429  9773.847427
   19        78.000000  9688.591224
   20        77.760000  9729.088064
   21        77.920000  9685.578117
   22        77.897143  9686.256162
   23        78.091429  9730.519230
   24        78.057143  9687.345697
   25        77.885714  9684.384937
   26        77.954286  9679.954080
   27        77.954286  9679.056792
   28        78.097143  9777.354926
   29        78.062857  9693.615713
   30        77.965714  9677.850507
   31        78.028571  9681.904210
   32        78.085714  9712.738997
   33        77.920000  9680.032112
   34        77.902857  9675.044482
   35        77.920000  9678.983614
   36        77.914286  9680.505580
   37        77.771429  9712.027397
   38        77.902857  9672.962454
   39        77.954286  9672.531875
   40        77.897143  9674.951230
   41        77.902857  9671.636640
   42        78.011429  9676.973853
   43        77.902857  9670.777098
   44        77.954286  9670.676686
   45        77.931429  9674.977301
   46        78.062857  9685.341618
   47        77.960000  9669.960712
   48        77.897143  9668.776380
   49        78.000000  9672.027343
```

50	77.902857	9668.312586
51	77.965714	9667.680120
52	78.040000	9675.842804
53	77.960000	9668.996584
54	77.902857	9674.040287
55	77.885714	9678.007836
56	77.960000	9667.931944
57	77.902857	9666.277168
58	78.000000	9670.411402
59	77.897143	9665.700915
60	77.640000	9723.965642
61	77.902857	9665.617558
62	77.811429	9685.405779
63	77.965714	9665.022534
64	77.960000	9665.027452
65	77.931429	9667.729257
66	77.965714	9664.332587
67	77.925714	9665.078710
68	77.902857	9664.412804
69	77.902857	9664.170384
70	77.937143	9665.258889
71	77.920000	9664.684560
72	77.828571	9680.629789
73	77.965714	9663.514359
74	77.965714	9664.772620
75	77.954286	9663.896912
76	78.005714	9669.700479
77	77.902857	9671.952375
78	77.902857	9662.291362
79	77.920000	9663.115547
80	77.977143	9665.008021
81	77.965714	9662.506421
82	77.965714	9663.912409
83	77.960000	9661.726460
84	77.965714	9663.277520
85	77.902857	9661.649374
86	77.937143	9661.291036
87	78.074286	9690.971838
88	77.902857	9661.052561
89	77.925714	9661.711118
90	77.925714	9664.546803
91	77.960000	9660.926144
92	77.965714	9661.186548
93	77.965714	9661.082036
94	77.965714	9660.987954

In [65]:
```
plt.plot(train_acc1[:,0], train_acc1[:,2])
plt.xlabel('Eopch')
plt.ylabel('Training loss')
```

Out[65]:
```
Text(0, 0.5, 'Training loss')
```



In [66]:
```
report1 = [{'Best learning rate':Best_lr, 'Best sigma2':Best_sigma2, 'Best cross
            'Best epoch':best_epoch,
            'Train accuracy (%)':train_acc1[best_epoch-1][1],
            'Test accuracy (%)':test_acc}]

report1 = pd.DataFrame.from_records(report1)
print(report1.to_string(index = False))
```

```
 Best learning rate  Best sigma2  Best cross val. acc. (%)  Best epoch  Train a
ccuracy (%)  Test accuracy (%)
              0.001        100.0                 78.097803          29
78.097143          78.888889
```

In [67]:
```
pred2 = prediction (Eval_misc,  w[best_epoch-1][:], b[best_epoch-1])
#print(pred1)
pred2.to_csv ('lr_misc_labels.csv', index = False, header=True)
```