Neural Network:

Importing panda and numpy:

In [5]:
```
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 [6]:
```
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 [7]:
```
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
```

Importing the tfidf datasets:

In [8]:
```
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 [9]:
```
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 [10]:
```
database = pd.concat([misc_train, misc_test, misc_eval], axis=0)
```

In [7]:
```
database
```

Out[7]:

	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 [8]:
```
database.dtypes
```

Out[8]:
```
defendant_age          object
defendant_gender       object
num_victims             int64
victim_genders         object
offence_category       object
offence_subcategory    object
dtype: object
```

In [11]:
```
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[11]:

	defendant_age	defendant_gender	num_victims	victim_genders	offence_category	offence_su
0	62.0	female	1	male	theft	theft
1	17.0	male	1	male	theft	po
2	0.0	male	1	male	theft	po
3	0.0	male	1	male	theft	sim
4	52.0	male	1	female	theft	po
...	...	...	...	...	...	
5245	0.0	male	1	male	theft	theft
5246	0.0	male	0	no_gender	sexual	
5247	0.0	male	1	male	theft	stealingF
5248	26.0	male	1	male	theft	
5249	16.0	male	1	female	theft	sim

25000 rows × 6 columns

In [12]:
```
# 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.
```

In [23]:
```
misc_transfered
```

Out[23]:

	defendant_age	num_victims	female	indeterminate	male	female	female;female	female;fem
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	
...	...	...	...	...	...	...	...	
5245	0.0	1	0	0	1	0	0	
5246	0.0	0	0	0	1	0	0	
5247	0.0	1	0	0	1	0	0	
5248	26.0	1	0	0	1	0	0	
5249	16.0	1	0	0	1	1	0	

25000 rows × 139 columns

In [13]:
```
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 [14]:
```
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)
```

In [30]:
```
print(Train_misc)
```

```
          1    2    3    4    5    6    7    8    9    10  ...  131  132  133  \
0      62.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0   
1      17.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0   
2       0.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0   
3       0.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  ...  1.0  0.0  0.0   
4      52.0  1.0  0.0  0.0  1.0  1.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0   
...     ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...   
17495   0.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0   
17496   0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0   
17497   0.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0   
17498   0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0   
17499   0.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0   

       134  135  136  137  138  139  140  
0      0.0  1.0  0.0  0.0  0.0  0.0 -1.0  
1      0.0  0.0  0.0  0.0  0.0  0.0 -1.0  
2      0.0  0.0  0.0  0.0  0.0  0.0  1.0  
3      0.0  0.0  0.0  0.0  0.0  0.0  1.0  
4      0.0  0.0  0.0  0.0  0.0  0.0 -1.0  
...    ...  ...  ...  ...  ...  ...  ...  
17495  0.0  1.0  0.0  0.0  0.0  0.0  1.0  
17496  0.0  0.0  0.0  0.0  0.0  0.0  1.0  
17497  0.0  0.0  0.0  0.0  0.0  0.0  1.0  
17498  0.0  0.0  0.0  0.0  0.0  0.0 -1.0  
17499  0.0  0.0  0.0  0.0  0.0  0.0  1.0  

[17500 rows x 140 columns]
```

In [15]:
```
# 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 [21]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score
```

In [39]:
```
Train_misc1 = Train_misc.to_numpy()
Train_misc_x = Train_misc1[:,:-1]
Train_misc_y = Train_misc1[:,-1]
Test_misc1 = Test_misc.to_numpy()
Test_misc_x = Test_misc1[:,:-1]
Test_misc_y = Test_misc1[:,-1]
Eval_misc1 = Eval_misc.to_numpy()
Eval_misc_x = Eval_misc1[:,:-1]
Eval_misc_y = Eval_misc1[:,-1]
```

In [40]:
```
predictions = mlp.predict(Test_misc_x)
```

In [ ]:
```
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
```

In [44]:
```


predictions_train = mlp.predict(Train_misc_x)
print(accuracy_score(predictions_train, Train_misc_y))
predictions_test = mlp.predict(Test_misc_x)
print(accuracy_score(predictions_test, Test_misc_y))
```

```
0.7976
0.4888888888888889
```

Cross validation procedure:

In [30]:
```
def cross_val(f1, f2, f3, f4, f5, hiddenLayers, learningRate):
    """
    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 = []
        train_data1 = train_data.to_numpy()
        valid_data1 = valid_data.to_numpy()
        Train_x = train_data1[:,:-1]
        Train_y = train_data1[:,-1]
        val_x = valid_data1[:,:-1]
        val_y = valid_data1[:,-1]

        mlp = MLPClassifier(solver='adam', hidden_layer_sizes=hiddenLayers, rand
                            learning_rate="invscaling",learning_rate_init = lear
        #print(train_data)
        mlp.fit(Train_x,Train_y)
        predictions = mlp.predict(val_x)
        ac = accuracy_score(predictions, val_y)
        acc.append (ac)
    Mean_acc = np.mean(acc)

    return Mean_acc
```

In [32]:
```
# Evaluating the network accuracy based on different values for learning rates ar
"""
"""
learning_rates = [0.1,1e-2,1e-3,1e-4]
hiddenLayers = [(10), (50), (100), (10,10), (50,50), (10,10,10), (50,50,50)]
#hiddenLayers = [(10)]

acc_mean = []
loss_mean = []
result = []


for lr in learning_rates:
    for hl in hiddenLayers:
        mean_ac = cross_val(f1, f2, f3, f4, f5, hl, lr)
        #print(mean_ac)
        acc_mean.append(mean_ac)
        result.append([lr, hl, mean_ac])
        #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]

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

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, 'hiddenLayer':Best_c, 'Best accuracy':
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
          0.1            10       0.78035
          0.1            50      0.781378
          0.1           100       0.78435
          0.1      (10, 10)       0.77812
          0.1      (50, 50)      0.779149
          0.1  (10, 10, 10)      0.781436
          0.1  (50, 50, 50)      0.782522
         0.01            10      0.787494
         0.01            50      0.783379
         0.01           100      0.784293
         0.01      (10, 10)      0.786523
         0.01      (50, 50)      0.784922
         0.01  (10, 10, 10)      0.786922
         0.01  (50, 50, 50)      0.784065
        0.001            10      0.787494
```

```
        0.001            50      0.783893
        0.001           100      0.784351
        0.001      (10, 10)      0.785779
        0.001      (50, 50)      0.782578
        0.001  (10, 10, 10)      0.786237
        0.001  (50, 50, 50)      0.784065
       0.0001            10      0.781207
       0.0001            50      0.789094
       0.0001           100      0.787951
       0.0001      (10, 10)       0.78738
       0.0001      (50, 50)      0.787894
       0.0001  (10, 10, 10)      0.788694
       0.0001  (50, 50, 50)       0.78738
Best learning rate: 0.0001
Best Cost: 50
 Best learning rate  Best Loss tradeoff  Best accuracy
             0.0001                  50       0.789094
```

In [40]:
```python
mlp = MLPClassifier(solver='adam', hidden_layer_sizes=Best_c, random_state=1, ma
                            learning_rate="invscaling",learning_rate_init = Best_
mlp.fit(Train_misc_x,Train_misc_y)
```

Out[40]:
```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=50, learning_rate='invscaling',
              learning_rate_init=0.0001, max_iter=1000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=1, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
```

In [42]:
```python
prediction_train = mlp.predict(Train_misc_x)
train_ac = accuracy_score(prediction_train, Train_misc_y)

prediction_test = mlp.predict(Test_misc_x)
test_ac = accuracy_score(prediction_test, Test_misc_y)
print(train_ac, test_ac)
```

```
0.7909142857142857 0.7982222222222223
```

In [60]:
```python
prediction_eval = mlp.predict(Eval_misc_x)
prediction_eval[prediction_eval==-1] = 0
#print(prediction_eval.shape)
pred =[]
for i in range (len(prediction_eval)):
    pred.append([i, prediction_eval[i]])

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

In [61]:
```python
#pred1 = prediction (Eval_misc,  w1[best_epoch-1][:], b1[best_epoch-1])
#print(pred1)
Pred.to_csv ('nn_misc_labels.csv', index = False, header=True)
```