CIFAR10_all_stages

May 16, 2018

1 PART1: CIFAR 10

```
In [ ]: import numpy as np
        import keras
        from keras.datasets import cifar10
        import matplotlib.pyplot as plt
        %matplotlib inline
        from sklearn.metrics import average_precision_score
        from __future__ import print_function
        from keras.models import Model, Sequential
        from keras.models import load_model #save and load models
        from keras import applications
        from keras import optimizers
        from keras.layers import Dense, Activation, Flatten
        from keras.layers import Conv2D, MaxPooling2D
        from sklearn import svm
        from sklearn.model_selection import KFold
        from sklearn.metrics import precision_recall_fscore_support
        import pickle
        from keras import regularizers

        import keras.backend as K
        K.clear_session()
```

```
In [ ]: #STAGE 1 starts here

        # The data, split between train and test sets
        (X_train, Y_train), (X_test, Y_test) = cifar10.load_data()
        #combining all the available data to use it in a way we want
        x = np.vstack((X_train,X_test))
        y = np.vstack((Y_train,Y_test))
        print('x shape:', x.shape)
        print('y shape:', y.shape)

        # normalize inputs from 0-255 to 0.0-1.0
        x = x.astype('float32')
        x = x/255
```

```
num_classes = 10
y1 = keras.utils.to_categorical(y, num_classes)
print('y1 shape', y1.shape)
print('Number of classes:', y1.shape[1])
```

In [ ]:
```
#Model parameters for target and shadow models
batch_size = 32 #upto us
epochs = 100
lrate = 0.001
decay = 1e-7 #find out what this decay parameter does
kernel_size = (5,5) #upto us
kernel_size2 = (3,3) #upto us
nout1 = 32 #upto us
nout2 = 32 #upto us
ndense = 128
#initializer in each layer - upto us
```

In [ ]:
```
data_size = [2500,5000,10000,15000]
target_rep = np.zeros((len(data_size),x.shape[0]))
ns = 10 #number of shadow models for one data_size

for i,ds in enumerate(data_size):
    sh = np.arange(x.shape[0])
    np.random.shuffle(sh)
    target_rep[i,:] = sh
    xtr_target = x[sh[:ds]]
    ytr_target = y1[sh[:ds]]
    xts_target = x[sh[ds:2*ds]]
    yts_target = y1[sh[ds:2*ds]]
    shadow_rep = np.zeros((ns,x.shape[0]-2*ds))
    sh2 = sh[2*ds:]

    #Training the target model when size of train & test data = ds
    model = Sequential()
    model.add(Conv2D(nout1, kernel_size,
                     padding='valid',
                     input_shape=xtr_target.shape[1:],
                     activation='tanh'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(nout2, kernel_size2, padding='valid', activation='tanh'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(ndense, activation='tanh'))
    model.add(Dense(num_classes, activation='softmax'))

    # initiate Adam optimizer
    opt = keras.optimizers.adam(lr=lrate, decay=decay)
```

```
# Let's train the model using Adam
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
#print model summary just once
if i == 0:
    print('Target model summary')
    print(model.summary())
# Fit the model
#put verbose = 0 when actually running
hist_target = model.fit(xtr_target, ytr_target,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=(xts_target, yts_target),
              shuffle=True,verbose=0)
print('\n\nFor target model with ds = %d'%ds)
print('Training accuracy = %f'%hist_target.history['acc'][-1])
print('Validation accuracy = %f'%hist_target.history['val_acc'][-1])
model_name = 'cifar10_target_'+str(ds)+'.h5'
model.save(model_name)
ytemp1 = model.predict(xtr_target)
ytemp2 = model.predict(xts_target)
xts_att = np.vstack((ytemp1,ytemp2))
yts_att = np.zeros(2*ds)
yts_att[:ds] = 1
xts_att_truelabels = np.vstack((ytr_target,yts_target))
xts_att_dict = {'xts_att':xts_att,'yts_att':yts_att,'xts_att_truelabels':xts_att_t
fname = './att_test_data_'+str(ds)
np.save(fname,xts_att_dict)

xtr_att = np.zeros((2*ds*ns,num_classes))
ytr_att = np.zeros((2*ds*ns,))
xtr_att_truelabels = np.zeros((2*ds*ns,num_classes))
for j in np.arange(ns):
    np.random.shuffle(sh2)
    shadow_rep[j,:] = sh2
    xtr_sh1 = x[sh2[:ds]]
    ytr_sh1 = y1[sh2[:ds]]
    xts_sh1 = x[sh2[ds:2*ds]]
    yts_sh1 = y1[sh2[ds:2*ds]]

    model_sh1 = Sequential()
    model_sh1.add(Conv2D(nout1, kernel_size,
                     padding='valid',
                     input_shape=xtr_sh1.shape[1:],
                     activation='tanh'))
    model_sh1.add(MaxPooling2D(pool_size=(2, 2)))
    model_sh1.add(Conv2D(nout2, kernel_size2, padding='valid', activation='tanh'))
```

```
                model_sh1.add(MaxPooling2D(pool_size=(2, 2)))
                model_sh1.add(Flatten())
                model_sh1.add(Dense(ndense, activation='tanh'))
                model_sh1.add(Dense(num_classes, activation='softmax'))

                # initiate Adam optimizer
                opt_sh1 = keras.optimizers.adam(lr=lrate, decay=decay)

                # Let's train the model using Adam
                model_sh1.compile(loss='categorical_crossentropy',
                                  optimizer=opt_sh1,
                                  metrics=['accuracy'])
                if j == 0 and i==0:
                    print('Shadow model summary:')
                    print(model_sh1.summary())
                hist_sh1 = model_sh1.fit(xtr_sh1, ytr_sh1,
                          batch_size=batch_size,
                          epochs=epochs,
                          validation_data=(xts_sh1, yts_sh1),
                          shuffle=True,verbose=0)
                model_name = 'cifar10_shadow_'+str(ds)+'_'+str(j)+'.h5'
                model_sh1.save(model_name)
                print('\nFor shadow model %d'%j)
                print('Training accuracy = %f'%hist_sh1.history['acc'][-1])
                print('Validation accuracy = %f'%hist_sh1.history['val_acc'][-1])
                ytemp11 = model_sh1.predict(xtr_sh1)
                ytemp22 = model_sh1.predict(xts_sh1)
                xtr_att[j*2*ds:(j+1)*2*ds] = np.vstack((ytemp11,ytemp22))
                ytr_att[j*2*ds:(2*j+1)*ds] = 1
                xtr_att_truelabels[j*2*ds:(j+1)*2*ds] = np.vstack((ytr_sh1,yts_sh1))

            #in outer for loop now
            datafile = './data_cifar10_shadow_'+str(ds)
            np.save(datafile,shadow_rep)
            xtr_att_dict = {'xtr_att':xtr_att,'ytr_att':ytr_att,'xtr_att_truelabels':xtr_att_t
            fname = './att_train_data_'+str(ds)
            np.save(fname,xtr_att_dict)
        #outside both for loops
        np.save('./data_cifar10_target',target_rep)
```

```
In [ ]: #STAGE 2 starts here
        def getfilename(datatype,ds):
            name = 'att_'+datatype+'_data_'+str(ds)+'.npy'
            return name
        def getmodelname(classi,ds):
            name = 'att_model_'+str(ds)+'_class_'+str(classi)+'.p'
            return name
```

```
C_test = [0.1,1,10]
gam_test = [0.001,0.01,0.1]
bestCgam = []
for ds in data_size:
    xtr_list = []
    ytr_list = []
    xts_list = []
    yts_list = []

    train_dict = np.load(getfilename('train',ds)).item()
    test_dict = np.load(getfilename('test',ds)).item()

    xtr_att_truelabels = train_dict['xtr_att_truelabels']
    xtr_att = train_dict['xtr_att']
    ytr_att = train_dict['ytr_att']

    xts_att_truelabels = test_dict['xts_att_truelabels']
    xts_att = test_dict['xts_att']
    yts_att = test_dict['yts_att']

    for i in np.arange(10):
        ind = np.where(xtr_att_truelabels[:,i]==1)[0]
        xtr_list.append(xtr_att[ind])
        ytr_list.append(ytr_att[ind])

        ind2 = np.where(xts_att_truelabels[:,i]==1)[0]
        xts_list.append(xts_att[ind2])
        yts_list.append(yts_att[ind2])

    class_c_g = np.zeros((10,2))
    prec = []
    recall = []
    acc_classwise = []
    yhat_full = np.zeros(2*ds)
    y_full = np.zeros(2*ds)
    start = 0
    for i in np.arange(10):
        x = xtr_list[i]
        y = ytr_list[i]

        xtest = xts_list[i]
        ytest = yts_list[i]
        ntest = xtest.shape[0]
        nfold = 5
        kf = KFold(n_splits=nfold, shuffle=True)
        acc = np.zeros((3,3,nfold))
        for ifold, ind in enumerate(kf.split(x)):
```

```
        # Get the training data in the split
        Itr,Its = ind
        xtr = x[Itr,:]
        ytr = y[Itr]
        xts = x[Its,:]
        yts = y[Its]
        for ic,c in enumerate(C_test):
            for ig,g in enumerate(gam_test):
                svc = svm.SVC(probability=False,  kernel="rbf", C=c, gamma=g,verbos
                svc.fit(xtr,ytr)
                yhat_ts = svc.predict(xts)
                acc[ic,ig,ifold] = np.mean(yhat_ts == yts)

    acc1 = np.mean(acc,axis=2)
    ci = np.argmax(np.amax(acc1,axis=1))
    gi = np.argmax(np.amax(acc1,axis=0))
    class_c_g[i,0] = C_test[ci]
    class_c_g[i,1] = gam_test[gi]

    #Now creating the actual attack classifier for class i and data_size ds
    svc = svm.SVC(probability=False,  kernel="rbf", C=C_test[ci], gamma=gam_test[g
    svc.fit(x,y)
    yhat_test = svc.predict(xtest)
    acci = np.mean(yhat_test == ytest)
    preci,reci,_,_= precision_recall_fscore_support(ytest,yhat_test,average='binar
    prec.append(preci)
    recall.append(reci)
    acc_classwise.append(acci)
    modelname = getmodelname(i,ds)
    with open( modelname, "wb" ) as fp:
        pickle.dump( [svc, C_test[ci], gam_test[gi]], fp)

    y_full[start:(start+ntest)] = ytest
    yhat_full[start:(start+ntest)] = yhat_test

    start = start+ntest

acctotal = np.mean(yhat_full == y_full)
prectotal,recalltotal,_,_= precision_recall_fscore_support(y_full,yhat_full,averag
fname = getfilename('results',ds)
dict_to_save = {'prectotal':prectotal,
               'acctotal':acctotal,
               'recalltotal':recalltotal,
               'prec':prec,
               'acc':acc_classwise,
               'recall':recall,
               'class_c_g':class_c_g}
np.save(fname,dict_to_save)
```

```
            print('For ds = ',ds)
            print('Attack Precision: ',prectotal)
            print('Attack Recall: ',recalltotal)
            print('Attack Accuracy: ',acctotal)
```

```
In [ ]: prec = []
        rec = []
        acc = []

        dict1 = np.load(getfilename('results',2500)).item()
        dict2 = np.load(getfilename('results',5000)).item()
        dict3 = np.load(getfilename('results',10000)).item()
        dict4 = np.load(getfilename('results',15000)).item()

        prec.append(dict1['prectotal'])
        prec.append(dict2['prectotal'])
        prec.append(dict3['prectotal'])
        prec.append(dict4['prectotal'])

        rec.append(dict1['recalltotal'])
        rec.append(dict2['recalltotal'])
        rec.append(dict3['recalltotal'])
        rec.append(dict4['recalltotal'])

        acc.append(dict1['acctotal'])
        acc.append(dict2['acctotal'])
        acc.append(dict3['acctotal'])
        acc.append(dict4['acctotal'])

        prec1 = dict1['prec']
        prec2 = dict2['prec']
        prec3 = dict3['prec']
        prec4 = dict4['prec']

        rec1 = dict1['recall']
        rec2 = dict2['recall']
        rec3 = dict3['recall']
        rec4 = dict4['recall']

        acc1 = dict1['acc']
        acc2 = dict2['acc']
        acc3 = dict3['acc']
        acc4 = dict4['acc']
        cgsvm1 = dict1['class_c_g']
        cgsvm2 = dict2['class_c_g']
        cgsvm3 = dict3['class_c_g']
        cgsvm4 = dict4['class_c_g']

        baseline = 0.5*np.ones(10)
        plt.plot(prec1)
        plt.plot(prec2)
        plt.plot(prec3)
        plt.plot(prec4)
        plt.plot(baseline)
        plt.xlabel('Classes')
        plt.ylabel('Precision')
        plt.legend(['2500','5000','10000','15000','baseline'],loc='lower right')
        plt.title('CIFAR-10, CNN , Membership Iinference Attack')


        plt.figure()
        plt.plot(data_size,prec)

        plt.ylabel('Precision')
        plt.xlabel('Training Set Size')
        plt.title('CIFAR-10, CNN , Membership Iinference Attack')
```

```
In [ ]: #MITIGATION PART
        reg = 5e-3
        print('Using L2 regularization with regularization constant = ',reg)
        data_size = [2500,5000,15000]
        target_rep = np.load('data_cifar10_target.npy')
        for i,ds in enumerate(data_size):
            sh = target_rep[i,:].astype(int)
            xtr_target = x[sh[:ds]]
            ytr_target = y1[sh[:ds]]
            xts_target = x[sh[ds:2*ds]]
            yts_target = y1[sh[ds:2*ds]]

            #Training the target model when size of train & test data = ds
            model = Sequential()
            model.add(Conv2D(nout1, kernel_size,
                             padding='valid',
                             input_shape=xtr_target.shape[1:],
                             activation='tanh',bias_regularizer=regularizers.l2(reg),kernel_reg
            model.add(MaxPooling2D(pool_size=(2, 2)))
            model.add(Conv2D(nout2, kernel_size2, padding='valid', activation='tanh',bias_regul
                             kernel_regularizer=regularizers.l2(reg)))
            model.add(MaxPooling2D(pool_size=(2, 2)))
            model.add(Flatten())
            model.add(Dense(ndense, activation='tanh',bias_regularizer=regularizers.l2(reg),
                            kernel_regularizer=regularizers.l2(reg)))
            model.add(Dense(num_classes, activation='softmax',bias_regularizer=regularizers.l2
                            kernel_regularizer=regularizers.l2(reg)))

            # initiate Adam optimizer
            opt = keras.optimizers.adam(lr=lrate, decay=decay)

            # Let's train the model using Adam
            model.compile(loss='categorical_crossentropy',
                          optimizer=opt,
                          metrics=['accuracy'])
            #print model summary just once
            if i == 0:
                print('Target model summary')
                print(model.summary())
            # Fit the model
            #put verbose = 0 when actually running
            hist_target = model.fit(xtr_target, ytr_target,
                          batch_size=batch_size,
                          epochs=epochs,
                          validation_data=(xts_target, yts_target),
                          shuffle=True,verbose=0)
            print('\n\nFor target model with ds = %d'%ds)
            print('Training accuracy = %f'%hist_target.history['acc'][-1])
            print('Validation accuracy = %f'%hist_target.history['val_acc'][-1])
            model_name = 'cifar10_target_after_mitigation_'+str(ds)+'.h5'
            model.save(model_name)
            ytemp1 = model.predict(xtr_target)
            ytemp2 = model.predict(xts_target)
            xts_att = np.vstack((ytemp1,ytemp2))
            yts_att = np.zeros(2*ds)
            yts_att[:ds] = 1
            xts_att_truelabels = np.vstack((ytr_target,yts_target))
            xts_att_dict = {'xts_att':xts_att,'yts_att':yts_att,'xts_att_truelabels':xts_att_t
            fname = './att_test_data_after_mitigation_'+str(ds)
            np.save(fname,xts_att_dict)
```

```
In [ ]: ## ATTACK RESULTS AFTER MITIGATION EMPLOYED IN TARGET MODEL
        for ds in data_size:
            xtr_list = []
            ytr_list = []
            xts_list = []
            yts_list = []

            train_dict = np.load(getfilename('train',ds)).item()
            test_dict = np.load(getfilename('test',ds)).item()
```

```
xtr_att_truelabels = train_dict['xtr_att_truelabels']
xtr_att = train_dict['xtr_att']
ytr_att = train_dict['ytr_att']

xts_att_truelabels = test_dict['xts_att_truelabels']
xts_att = test_dict['xts_att']
yts_att = test_dict['yts_att']

for i in np.arange(10):
    ind = np.where(xtr_att_truelabels[:,i]==1)[0]
    xtr_list.append(xtr_att[ind])
    ytr_list.append(ytr_att[ind])

    ind2 = np.where(xts_att_truelabels[:,i]==1)[0]
    xts_list.append(xts_att[ind2])
    yts_list.append(yts_att[ind2])

class_c_g = np.zeros((10,2))
prec = []
recall = []
acc_classwise = []
yhat_full = np.zeros(2*ds)
y_full = np.zeros(2*ds)
start = 0
for i in np.arange(10):
    x = xtr_list[i]
    y = ytr_list[i]

    xtest = xts_list[i]
    ytest = yts_list[i]
    ntest = xtest.shape[0]
    nfold = 5
    kf = KFold(n_splits=nfold, shuffle=True)
    acc = np.zeros((3,3,nfold))
    for ifold, ind in enumerate(kf.split(x)):
        # Get the training data in the split
        Itr,Its = ind
        xtr = x[Itr,:]
        ytr = y[Itr]
        xts = x[Its,:]
        yts = y[Its]
        for ic,c in enumerate(C_test):
            for ig,g in enumerate(gam_test):
                svc = svm.SVC(probability=False,  kernel="rbf", C=c, gamma=g,verbos
                svc.fit(xtr,ytr)
                yhat_ts = svc.predict(xts)
                acc[ic,ig,ifold] = np.mean(yhat_ts == yts)
    acc1 = np.mean(acc,axis=2)
    ci = np.argmax(np.amax(acc1,axis=1))
    gi = np.argmax(np.amax(acc1,axis=0))
    class_c_g[i,0] = C_test[ci]
    class_c_g[i,1] = gam_test[gi]

    #Now creating the actual attack classifier for class i and data_size ds
    svc = svm.SVC(probability=False,  kernel="rbf", C=C_test[ci], gamma=gam_test[g
    svc.fit(x,y)
    yhat_test = svc.predict(xtest)
    acci = np.mean(yhat_test == ytest)
    preci,reci,_,_= precision_recall_fscore_support(ytest,yhat_test,average='binar
    prec.append(preci)
    recall.append(reci)
    acc_classwise.append(acci)
    modelname = getmodelname(i,ds)
    with open( modelname, "wb" ) as fp:
        pickle.dump( [svc, C_test[ci], gam_test[gi]], fp)

    y_full[start:(start+ntest)] = ytest
    yhat_full[start:(start+ntest)] = yhat_test

    start = start+ntest

acctotal = np.mean(yhat_full == y_full)
prectotal,recalltotal,_,_= precision_recall_fscore_support(y_full,yhat_full,averag
fname = getfilename('results',ds)
dict_to_save = {'prectotal':prectotal,
               'acctotal':acctotal,
               'recalltotal':recalltotal,
               'prec':prec,
               'acc':acc_classwise,
               'recall':recall,
               'class_c_g':class_c_g}
np.save(fname,dict_to_save)
print('For ds = ',ds)
print('Attack Precision: ',prectotal)
print('Attack Recall: ',recalltotal)
print('Attack Accuracy: ',acctotal)
```

2 PART2: UCI ADULT (CENSUS INCOME)

```
In [ ]: df = pd.read_csv("adult census/adult.data", names=[
                "Age", "Workclass", "fnlwgt", "Education", "Education-Num", "Martial Status",
                "Occupation", "Relationship", "Race", "Sex", "Capital Gain", "Capital Loss",
                "Hours per week", "Country", "Target"],
                sep=r'\s*,\s*',
                engine='python',
                na_values="?")
        df1 = pd.read_csv(
            "adult census/adult.test",
            names=[
                "Age", "Workclass", "fnlwgt", "Education", "Education-Num", "Martial Status",
                "Occupation", "Relationship", "Race", "Sex", "Capital Gain", "Capital Loss",
                "Hours per week", "Country", "Target"],
                sep=r'\s*,\s*',
                engine='python',
                na_values="?")
        data = pd.concat([df, df1], ignore_index= True)
        print(data.shape)
        print(data.dtypes)
```

```
In [ ]: df1.head()
```

```
In [ ]: names_cloud = data.columns.tolist()
        print(names_cloud)
        X = np.array(data[names_cloud])
        print(X.shape)
```

```
In [ ]: fig = plt.figure(figsize=(20,20))
        cols = 5
        rows = (float(data.shape[1]) / cols)
        for i, column in enumerate(data.columns):
            a = fig.add_subplot(rows, cols, i + 1)
            a.set_title(column)
            if data.dtypes[column] == np.object:
                data[column].value_counts().plot(kind="bar", axes=a)
            else:
                data[column].hist(axes=a)
                plt.xticks(rotation="vertical")
        plt.subplots_adjust(hspace=0.7, wspace=0.2)
```

```
In [ ]: y = (data['Target'].map({"<=50K":0,">50K":1})).values
        print(pd.value_counts(pd.Series(y)))
        data.drop('Target',axis=1, inplace =True,)
```

```
In [ ]: categorical_features = data.select_dtypes(include=['object']).columns
        print(categorical_features)
        ohc_category = ['Workclass', 'Education', 'Martial Status', 'Occupation', 'Relationshi
        df_ohc = pd.get_dummies(data, columns = ohc_category)
        print(df_ohc.shape)
        df_ohc.head()
```

```
In [ ]: names_x = df_ohc.columns.tolist()
        print("Target Variable: Target")
        print("Predictors: "+str(names_x))
        x = np.array(df_ohc[names_x])
        print("Number of data samples : {0:d}".format(x.shape[0]))
        print("Number of Predictor Features : {0:d}".format(x.shape[1]))
```

```
In [ ]: x = x.astype('float32')
        x = x/255
        batch_size = 32 #upto us
        epochs = 100
        lrate = 0.001
        decay = 1e-7
        data_size = 10000
        ns = 20 #number of shadow models for one data_size
        nh = 5 #number of hidden layers
        nout = 1
        seed = 7
        np.random.seed(seed)
        sh = np.arange(x.shape[0])
        np.random.shuffle(sh)
        target_rep = np.zeros((1,x.shape[0]))
        target_rep[0,:] = sh
        print(sh)
```

```
In [ ]: k.clear_session()
        xtr_target = x[sh[:data_size]]
        ytr_target = y[sh[:data_size]]
        xts_target = x[sh[data_size:data_size*2]]
        yts_target = y[sh[data_size:2*data_size]]
        shadow_rep = np.zeros((20,x.shape[0]-2*data_size))
        sh1 = sh[2*data_size:]
        xtr_att = np.zeros((2*data_size*ns,1))
        ytr_att = np.zeros((2*data_size*ns,1))
        xtr_att_truelabels = np.zeros((2*data_size*ns,))
        model_target = Sequential()
        model_target.add(Dense(nh, input_shape =(x.shape[1],), activation='sigmoid', name = 'hi
        model_target.add(Dense(1, activation='sigmoid', name = 'output'))
        opt = keras.optimizers.adam(lr=lrate, decay=decay)
        model_target.compile(loss='binary_crossentropy',
                          optimizer=opt,
                          metrics=['accuracy'])
        print(model_target.summary())
        hist_target = model_target.fit(xtr_target, ytr_target,
                          batch_size = batch_size,
                          epochs = epochs,
                          validation_data=(xts_target, yts_target), shuffle=True, verbose=0)
        print('\n\nFor target model with training datasize = %d'%data_size)
        print('Training accuracy = %f'%hist_target.history['acc'][-1])
        print('Validation accuracy = %f'%hist_target.history['val_acc'][-1])
        model_target_name = 'UCI_Adult_target_'+str(data_size)+'.h5'
        model_target.save(model_target_name)
        ytemp_tr_target = model_target.predict(xtr_target)
        ytemp_ts_target = model_target.predict(xts_target)
        xts_att = np.vstack((ytemp_tr_target,ytemp_ts_target))
        yts_att = np.zeros((2*data_size,1))
        yts_att[data_size:2*data_size] = 1
        xts_att_truelabels = np.vstack((ytr_target,yts_target))
        xts_att_dict = {'xts_att':xts_att,'yts_att':yts_att,'xts_att_truelabels':xts_att_truela
        fname = './att_test_data_'+str(data_size)
        np.save(fname,xts_att_dict)
        datafile = './data_adult_target_'+str(data_size)
        np.save(datafile,target_rep)

In [ ]: for i in np.arange(ns):
            np.random.shuffle(sh1)
            shadow_rep[i,:] = sh1
            xtr_shadow = x[sh1[:data_size]]
            ytr_shadow = y[sh1[:data_size]]
            xts_shadow = x[sh1[data_size:2*data_size]]
            yts_shadow = y[sh1[data_size:2*data_size]]
            model_shadow = Sequential()
            model_shadow.add(Dense(nh, input_shape =(x.shape[1],), activation='sigmoid', name =
            model_shadow.add(Dense(1, activation='sigmoid', name = 'output'))
            opt = keras.optimizers.adam(lr=lrate, decay=decay)
            model_shadow.compile(loss='binary_crossentropy',
                          optimizer=opt,
                          metrics=['accuracy'])
            if i == 0:
                print("Shadow Model Summary")
                print(model_shadow.summary())
            hist_shadow = model_shadow.fit(xtr_shadow, ytr_shadow,
                          batch_size = batch_size,
                          epochs = epochs,
                          validation_data=(xts_shadow, yts_shadow), shuffle=True, verbose=0)
            print("Shadow model no: %d"%i)
            print('\n\nFor shadow model with training datasize = %d'%data_size)
            print('Training accuracy = %f'%hist_shadow.history['acc'][-1])
            print('Validation accuracy = %f'%hist_shadow.history['val_acc'][-1])
            ytemp11 = model_shadow.predict(xtr_shadow)
            ytemp22 = model_shadow.predict(xts_shadow)
            model_shadow_name = 'UCI_Adult_shadow_'+str(data_size)+'_'+str(i)+'.h5'
            print(model_shadow_name)
            model_shadow.save(model_shadow_name)
            xtr_att[i*2*data_size:(i+1)*2*data_size] = np.vstack((ytemp11,ytemp22))
            ytr_att[((i*2)+1)*data_size:(i+1)*2*data_size] = 1
            xtr_att_truelabels[i*2*data_size:(i+1)*2*data_size] = np.hstack((ytr_shadow,yts_sha
        datafile = './data_adult_shadow_'+str(data_size)
        np.save(datafile,shadow_rep)
        xtr_att_dict = {'xtr_att':xtr_att,'ytr_att':ytr_att,'xtr_att_truelabels':xtr_att_truel
        fname = './att_train_data_'+str(data_size)
        np.save(fname,xtr_att_dict)
```

```
In [ ]: model_attack = Sequential()
        model_attack.add(Dense(nh, input_shape = (xtr_att.shape[1],), activation='sigmoid', na
        model_attack.add(Dense(1, activation='sigmoid', name = 'output'))
        opt = keras.optimizers.adam(lr = lrate, decay=decay)
        model_attack.compile(loss='binary_crossentropy',
                          optimizer=opt,
                          metrics=['accuracy'])
        print("Attack Model Summary")
        print(model_attack.summary())
        hist_attack = model_attack.fit(xtr_att, ytr_att,
                          batch_size = batch_size,
                          epochs = epochs,
                          validation_data=(xts_att, yts_att), shuffle=True, verbose=0)
        print('\n\nFor attack model with training datasize = %d'%xtr_att.shape[0])
        print('Training accuracy = %f'%hist_attack.history['acc'][-1])
        print('Validation accuracy = %f'%hist_attack.history['val_acc'][-1])
        y_score = model_attack.predict(xts_att)
        average_precision = average_precision_score(yts_att, y_score)
        print('Average precision-recall score: {0:0.2f}'.format(
              average_precision))
```

3 Mitigation Strategy

```
In [ ]: data_size = [2500,5000,15000]
        num = 0
        for ds in data_size:
            xtr_list = []
            ytr_list = []
            xts_list = []
            yts_list = []
            name = 'att_test_data_after_mitigation_' +str(ds)+'.npy'
            test_dict = np.load(name,encoding='bytes').item()


            xts_att_truelabels = test_dict[b'xts_att_truelabels']
            xts_att = test_dict[b'xts_att']
            yts_att = test_dict[b'yts_att']

            for i in np.arange(10):
                ind2 = np.where(xts_att_truelabels[:,i]==1)[0]
                xts_list.append(xts_att[ind2])
                yts_list.append(yts_att[ind2])
```

```
    prec = []
    recall = []
    acc_classwise = []
    yhat_full = np.zeros(2*ds)
    y_full = np.zeros(2*ds)
    start = 0
    for i in np.arange(10):
        xtest = xts_list[i]
        ytest = yts_list[i]
        ntest = xtest.shape[0]
        name = 'att_model_'+str(ds)+'_class_'+str(i)+'.p'
        with open( name, "rb" ) as fp:
            svc, _,_ = pickle.load(fp)
        yhat_test = svc.predict(xtest)
        acci = np.mean(yhat_test == ytest)
        preci,reci,_,_= precision_recall_fscore_support(ytest,yhat_test,average='binar
        prec.append(preci)
        recall.append(reci)
        acc_classwise.append(acci)
        y_full[start:(start+ntest)] = ytest
        yhat_full[start:(start+ntest)] = yhat_test

        start = start+ntest

    acctotal = np.mean(yhat_full == y_full)
    prectotal,recalltotal,_,_= precision_recall_fscore_support(y_full,yhat_full,averag
    print('For ds = ',ds)
    print('Attack Precision: ',prectotal)
    print('Attack Recall: ',recalltotal)
    print('Attack Accuracy: ',acctotal)
    strname = 'Accuracy vs classes curve after applying mitigation'

    plt.plot(acc_classwise)

plt.title(strname)
plt.xlabel('Classes')
plt.ylabel('Accuracy')
plt.legend(['2500','5000','15000'])
```