# Mining Advisor-advisee Relationships

1. Introduction

This Lab focuses on learning basic machine learning methods and implementing them on a specific topic, to find advisor- advisee relationships in academic heterogeneous networks. In this Lab, you will learn some machine learning tools and realize your own model based on Python and Sklearn. Moreover, you will use Keras and Tensorflow to build a deep-learning model and compare the performance between deep learning and traditional machine learning methods.

1. Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of [TensorFlow](https://github.com/tensorflow/tensorflow), [CNTK](https://github.com/Microsoft/cntk), or [Theano](https://github.com/Theano/Theano). It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Keras has the following advantages:

Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).

Supports both convolutional networks and recurrent networks, as well as combinations of the two.

Runs seamlessly on CPU and GPU.

1. Instruction

In this experiment, we have the known (GroundTruth) Advisor-advisee Relationships (AARs) and common coauthor relationships, which are obtained through calculating the probability of being an AAR respectively. The authors are represented by eight 4-bit code.

We have extracted features in the perspective of relationship before, in and outside one cooperation. For instance, if we have known A cooperated with B in paper publication from 2008, then the more paper A has before 2008 than B, the more likely that A is Advisor and B is Advisee in their cooperation. We have 22 ordered features ranked through Mutual Information Correlations, method of feature engineering (realized in Python sklearn).

The features of the known AARs are used to train and test our machine learning model and we will use the trained model to predict if the common coauthor relationships are AAR or not. If interested, you can compare the performance with Decision Tree and SVM.

1. Project Source Code

# coding: utf-8

"""

Machine Learning Model for Predicting Advisor-Advisee Relationship.

"""

import tensorflow as tf

from keras.models import Sequential, Model

from keras.layers import Dense, Activation, Input

from keras.metrics import \*

import matplotlib.pyplot as plt

#import seaborn as sns

import numpy as np

from keras.utils import np\_utils

import pandas as pd

from time import time

from sklearn import metrics

from sklearn.svm import SVC

from sklearn import tree

from time import time

#prepare dataset

raw\_data = open('GroundTruth\_and\_Features.csv','r')

# load the CSV file as a numpy matrix

FeatureAmount = 22

dataset = np.loadtxt(raw\_data, delimiter=",")[:,:FeatureAmount+1]

print ("dataset2: ",dataset[:5])

# separate the data from the target attributes

X = dataset[0:,1:]

y = dataset[0:,0]

X\_train = dataset[0:-100000,1:]

y\_train = dataset[0:-100000,0]

X\_test = dataset[-100000:,1:]

y\_test = dataset[-100000:,0]

shape\_in = X\_train.shape[1]

shape\_out = 2

#construct Machine Learning model

def one\_hot\_encode\_object\_array(arr):

'''One hot encode a numpy array of objects (e.g. strings)'''

uniques, ids = np.unique(arr, return\_inverse=True)

return np\_utils.to\_categorical(ids, len(uniques))

y\_train\_ohe = one\_hot\_encode\_object\_array(y\_train)

y\_test\_ohe = one\_hot\_encode\_object\_array(y\_test)

def SVM(X\_train,y\_train,X\_test,y\_test):

# fit a SVM model to the data

t0 = time()

model = SVC(kernel = "linear")

model.fit(X\_train, y\_train)

print ("training time:", round(time()-t0, 3), "s")

#print(model)

# make predictions

t0 = time()

expected = y\_test

predicted = model.predict(X\_test)

print ("predicting time:", round(time()-t0, 3), "s")

# summarize the fit of the model

score = metrics.accuracy\_score(expected, predicted)

print(score)

print(metrics.recall\_score(expected,predicted))

return model,score

#print (data.head())

def DTree(X\_train,y\_train,X\_test,y\_test):

model = tree.DecisionTreeClassifier(min\_samples\_split = 40)

t0 = time()

model.fit(X\_train,y\_train)

print ("training time:", round(time()-t0, 3), "s")

t0 = time()

expected = y\_test

predicted = model.predict(X\_test)

print ("predicting time:", round(time()-t0, 3), "s")

# summarize the fit of the model

score = metrics.accuracy\_score(expected, predicted)

print(score)

print(metrics.recall\_score(expected,predicted))

return model,score

def simple\_deep\_learning\_model(X\_train,y\_train,X\_test,y\_test,X\_all,shape\_in):

model = Sequential()

model.add(Dense(5, input\_shape = (shape\_in,))) #input scale

model.add(Activation('sigmoid'))

#model.add(Dense(32, activation='sigmoid'))

#model.add(Activation('sigmoid'))

model.add(Dense(2)) #output scale

model.add(Activation('softmax'))

t0 = time()

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

model.fit(X\_train, y\_train, nb\_epoch=5, batch\_size=5, verbose=1)

print ("training time:", round(time()-t0, 3), "s")

loss, accuracy = model.evaluate(X\_test, y\_test, verbose=0)

print ("Accuracy = {:.5f}".format(accuracy))

print ("loss = {:.5f}".format(loss))

y\_pred\_test = model.predict(X\_test)

y\_pred\_all = model.predict(X\_all)

return model,y\_pred\_test,y\_pred\_all

model\_deep\_learning, y\_pred, y\_pred\_all = simple\_deep\_learning\_model(X\_train,y\_train\_ohe,X\_test,y\_test\_ohe,X,FeatureAmount)

def predict(model,dataset):

file\_rd = open(dataset,'r')

print("start loading...")

data = np.loadtxt(file\_rd, delimiter=",")[1:,:]

print("loading finished!")

predicted = model.predict(data)

return predicted

def print\_res(arr,output\_file):

file\_wrt = open(output\_file,'w')

for p in arr:

file\_wrt.write(str(p)+'\n')

def output\_prediction():

predicted = predict(model\_deep\_learning,'coauthor\_feature\_data.csv')

print("\n")

print("start switching")

predicted\_arr\_num = cal\_set\_num(predicted)

print("switching finished!")

print("start outputing")

print\_res(predicted\_arr\_num,'mentor\_unilateral\_result\_tmp.csv')

print("Outputing finished!")

file\_input\_author = open('coauthor\_feature\_raw.csv','r')

file\_input\_res = open('mentor\_unilateral\_result\_tmp.csv','r')

file\_output = open('MentorshipPredictionResult\_IDPair2Probability.csv','w')

count=0 # changed

while(1):

count+=1

if(count%1000000 == 0):

print (datetime.datetime.now(),count,"MentorData")

line1 = file\_input\_author.readline()

line2 = file\_input\_res.readline()

if not line1 or not line2:

break

line1 = line1.strip()

line1\_list = line1.split(',')

id\_i = line1\_list[0]

id\_j = line1\_list[1]

res = float(line2.strip())

file\_output.write(','.join(line1\_list[:2])+','+str(res)+'\n')

print("finish!")

def cal\_set\_num(pred\_set):

y\_pred\_arr\_num = []

for res in pred\_set:

y\_pred\_arr\_num.append(res[1])

return y\_pred\_arr\_num

output\_prediction()