陶文政

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.

2 Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or 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.

3 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.

4 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()
```