DL Project
Date: May 10th, 2020

Project Documentation

1. Environment

All programs are written in Python 3.7

1.1 run in shell (not recommended)

1.2 run in Jupiter Notebook with Anaconda 2020.02 (recommended)

Note: Anaconda includes all of the Python packages used frequently in ML/DL

2. Packages

2.1 Built-in packages included in Anaconda:

xlrd 1.2.0, seaborn 0.9.0 , scikit-learn 0.21.3, pandas 0.25.1,
zipp 0.6.0, numpy 1.17.2

2.2 Additional packages:

tensorflow 2.1.0, pandas 1.0.3, keras 2.3.1, gensim 3.8.0, , tabulate 0.8.6

3. Program Overview

The completed process is shown in diagram in section 5. The program contains python files which process the slice files, tokenize source codes, transform tokens to vectors, split the dataset, convert vectors to same-length vectors, fit the deep learning model, evaluate model, predict new data, and visualize deep learning output .

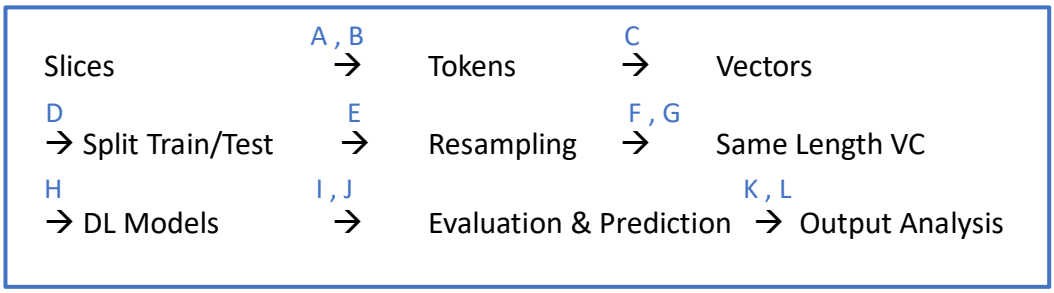
4. Pre-execution

Set up working directory, vulnerability type, random seed, paths used in all functions

Command:

```
import os
os.chdir("/Users/amy_a/Desktop/finalCodes")
vType = "ALL"
randomSeed = 1099
numSamples = 100 #Max Num of slice samples from each file
vectorDim = 30 #num of vector cols
slicePath = './data/slicesSource/'
tokenPath = './data/token/SARD/'
w2vmodelPath = './w2vModel/model/w2vModel_ALL'
vectorPath =  './data/vector/'
vectorTrainPath = './data/DLvectors/train/'
vectorTestPath = './data/DLvectors/test/'
dlInputsTrainPath = './data/DLinputs/train/'
dlInputsTestPath  = './data/DLinputs/test/'
```

5. Program Details



A. slicesToTokens.py

Aim: Transform slice files to "Token" .pkl files
Input: Raw Slice files.txt
Output: .pkl files of each program, containing 5 items in array
[list of tokens, label, function list in each program, filename, vulnerability type]
Output Files saved in './data/token/SARD'

Command:

```
from slicesToTokens import *
mycase_ID = tokenizeSlices(slicePath, tokenPath, numSamples)
```

B. isDuplicatedID.py

Aim: Check if there are any duplicates sample ID
Input: List of sample ID (all folder names in corpus folder)
Output: True (duplicated ID in .txt file)/ False (No duplicated ID)

Command:

```
from isDuplicatedID import *
print("The dataset has duplicated ID: ", isDuplicatedID(mycase_ID))
```

C. tokensToVectors.py

Aim: Transform tokens to vectors using the W2V model
Input: Tokens in index 0 in each .pkl file in './data/token/SARD'
Output:
1. W2V Model created and saved in './w2vModel/w2vModel_ALL'
2. Vocabs in W2V Model saved in wordsW2Vmodel.txt
3. Transformed tokens to vectors saved in './data/vector/'

For 1 program, vector array has 30 columns(Vdim), row = (#of tokens)

Command:

```
from tokensToVectors import *
myW2Vmodel = createW2VModel(w2vmodelPath, tokenPath, vectorDim)
fitW2VModel(w2vmodelPath, tokenPath, vectorPath)
```

D. splitTrainTest.py

Aim: Randomly Split dataset to Train/Test Set
Input: Vector files in './data/vector/'
Output: ALL_Train.pkl in './data/DLvectors/train/'
ALL_Test.pkl in './data/DLvectors/test/'

Command:

```
from splitTrainTest import *
splitTrainTest(vType, vectorPath, vectorTrainPath,
vectorTestPath,randomSeed, split = 0.8 )
```

E. downSampling.py

Aim: get the balanced train set of Class 0, 1
Input: Train set in train.pkl
Output: Balanced Train set in balancedClassTrain.pkl
Saved in ./data/DLvectors/train/

Command:

```
from downSampling import *
caseID_one,caseID_zero,downsampleNum = appendCaseIDLabel0(vectorTrainPath)
downsampling (caseID_one,caseID_zero, downsampleNum , randomSeed, vectorTr
ainPath)

#Optional used to check if the class label are balanced
print("Class Label is balanced: " , isClassBalanced(vectorTrainPath))
```

F. adjustVectorLen.py

Aim: calculate Mean Vector Length
transform each sample's "Vector Length" to Mean Length.
(row or num of tokens in each sample)
Input: Balanced Train & Test sets in ./data/DLvectors/
Output: Balanced Train & Test sets with same length vectors in ./data/DLInputs

Command:

```
from adjustVectorLen import *
avg = meanLen(vectorTrainPath)
tranformVectorLen(vectorTrainPath, vectorTestPath, dlInputsTrainPath,
dlInputsTestPath, avg, vectorDim, vType)
print("New Vector Length (rows x cols): " ,avg, " x " ,vectorDim)
```

G. saveKeyData.py

Aim: save case ID, Class label, vulnerability type from final train set to .txt file
Input: Balanced Final Train & Test Final sets in ./data/DLvectors/
Output: trainKeyData.txt, testKeyData.txt

Command:

```
from saveKeyData import *
saveKeyData(dlInputsTrainPath)
saveKeyData(dlInputsTestPath)
```

H. DLModel.py

Aim: customize, build BGRU model and fit it with train set
Input: final train set
Output: fitted BGRU model saved in './model/BRGU_ALL'

Command:

```
from DLModel import *
myoptimizer = 'adam' #can be changed to ‘adamax’
maxlen = avg #avg calculated from part 5.6
layers = 2
dropout = 0.2
batchSize = 32
vectorDim = 30
```

Part (a): fitting BGRU model

```
#Build BGRU Model with parameters
myKerasModel =  buildBGRU(maxlen, vectorDim, layers, dropout,myoptimizer )

#Fit BGRU Model with trained data and saved the model for later use
weightpath = './model/BRGU_ALL' + myoptimizer +str(randomSeed)
mymodel = fitModel(myKerasModel, weightpath, dlInputsTrainPath, batchSize,
maxlen, vectorDim, randomSeed)
```

Part (b): fitting BLSTM model

```
#Build BLSTMModel with parameters
myKerasModel =  buildBLSTM(maxlen, vectorDim, layers, dropout,myoptimizer )

#Fit BLSTM Model with trained data and saved the model for later use
weightpath = './model/BLSTM_ALL' + myoptimizer +str(randomSeed)
mymodel = fitModel(myKerasModel, weightpath, dlInputsTrainPath, batchSize,
maxlen, vectorDim, randomSeed)
```

I. ModelPrediction.py

Aim: load DL model and fit it with test set for model evaluation
Input: final test set and saved model
Output: output values and predicted values from Model saved to excel file (OutputSummary_adamRandomseed.xlsx)

Command:

Note: These parameters are same as section H and same for part (a) BGRU, part (b) BLSTM

```
#all parameters are same as section H
from DLModel import *
from DLPrediction import *
myoptimizer = 'adam'
maxlen = avg
layers = 2
dropout = 0.2
batchSize = 32
```

Part (a): Evaluate BGRU model and Predict output with Test set

```
modelName = 'BGRU'
weightpath = './model/BRGU_ALL' + myoptimizer +str(randomSeed)
myKerasModelADAM =  buildBGRU(maxlen,vectorDim, layers, dropout,myoptimizer )
myKerasModelADAM.load_weights(weightpath)
testID_label, output_dl_labels, mypredicted_labels, myreallabels,
myvtypelabels  = predictLabel(myKerasModelADAM, dlInputsTestPath, maxlen,
vectorDim, myoptimizer, modelName, randomSeed)
```

Part (b): Evaluate BGRU model and Predict output with Test set

```
modelName = 'BLSTM'
weightpath = './model/BLSTM_ALL' + myoptimizer +str(randomSeed)
myKerasModelADAM2 =  buildBLSTM(maxlen,vectorDim,layers,dropout,myoptimizer )
myKerasModelADAM2.load_weights(weightpath)
testID_label2, output_dl_labels2, mypredicted_labels2, myreallabels2,
myvtypelabels  = predictLabel(myKerasModelADAM2, dlInputsTestPath, maxlen,
vectorDim, myoptimizer, modelName, randomSeed)
```

J. ConfusionMatrix.py

Aim: get simple confusion matrix for model evaluation
Input: predicted label and real label from part I
Output: confusion matrix (TP/TN/FP/FN, Accuracy, Specificity, etc.)

Command:

```
from ConfusionMatrix import *
getConfusionMatrix(mypredicted_labels, myreallabels)#Part(a) BGRU
getConfusionMatrix(mypredicted_labels2, myreallabels2)#Part(b)BLSTM
```

K. evaluateModels.py

Aim: get simple confusion matrix for different prediction thresholds
Input: predicted label and real label from part I
Output: Table (TP/TN/FP/FN, Accuracy, Specificity, etc.) in each threshold

Command:

```
from evaluateModels import *
thresdArray = [0.40, 0.45,0.5, 0.53, 0.55, 0.58, 0.60, 0.65, 0.70, 0.8]
mydata, recall, precision, specificity, F1, Accuracy, balanceAccuracy  =
combinedPredictions(thresdArray, DLdata)
mydata.to_excel("predictionWithDiffThreshold_gruadam.xlsx")
```

L. plotCounts.py

Aim: plot histogram and bar chart corresponding to threshold
Input: predicted label and real label from part K
Output: histogram and bar (TP/TN/FP/FN)

Command:

```
from evaluateModels import *
thresdArray = [0.40, 0.45,0.5, 0.53, 0.55, 0.58, 0.60, 0.65, 0.70, 0.8]
mydata, recall, precision, specificity, F1, Accuracy, balanceAccuracy  =
combinedPredictions(thresdArray, DLdata)
mydata.to_excel("predictionWithDiffThreshold_gruadam.xlsx")
```

Appendix A : Plot Recall VS Precision

Appendix B : Plot F1, balancedAccuracy with Different Thresholds

Appendix C : Plot Accuracy, balancedAccuracy with Different Thresholds

Appendix D : Plot Accuracy, Specificity with Different Thresholds