**Problem Statement/ Scenario Description:** You are advised to design a Part-of-Speech (POS) tagger using Bidirectional Long Short Term Memory. The POS tagger should take a tokenized sentence as an input and produce POS tags for each token of the sentence as output.

Environment: TCS iON Virtual Environment is to be used.

Use the file “train.txt” as data source.

**Question**:

1. Use a Bi-LSTM to extract the hidden feature (concatenation of forward and backward) for each word.
2. Do Pre-processing and Embed each word with the corresponding pre-trained word embedding.
3. Employ a feed-forward network to classify each word to their corresponding POS tag from a tag set
4. Evaluation: Perform 3-fold cross-validation on the training dataset and report Overall precision, recall and F1-score; Tag-wise precision, recall and F1-score
5. Error Analysis: Confusion matrix (Each element Aij of matrix A denotes the number of times tag i classified as tag j; Statistics of the tag set

**Answer**:

#!/usr/bin/env python

# coding: utf-8

# In[1]:

# Import the required libraries.

import re

import math

import random

import collections

import operator

import numpy as np

import pandas as pd

from sklearn.model\_selection import KFold

from sklearn.metrics import precision\_recall\_fscore\_support, f1\_score

from sklearn.metrics import confusion\_matrix, classification\_report

from collections import defaultdict, Counter

import matplotlib

from matplotlib import pyplot as plt

import seaborn as sns

random.seed(11)

np.random.seed(11)

# In[2]:

from keras.models import Sequential

from keras.layers import Dense, LSTM, Conv1D, InputLayer, Bidirectional, TimeDistributed, Embedding, Activation, Masking

from keras.optimizers import Adam, SGD

from keras.preprocessing.sequence import pad\_sequences

# In[ ]:

def parse\_sentence(sentence):

'''

Function for parsing the words and tags from the

sentences of the input corpus.

'''

word\_tag\_pairs = sentence.split(" ")

words = []

tags = []

for i, word\_tag in enumerate(word\_tag\_pairs):

word, tag = word\_tag.strip().rsplit('/', 1)

words.append(word)

tags.append(tag)

return words, tags

# In[ ]:

#Loading the data and parsing the sentences into a list.

parsed\_sentences = []

with open('train.txt', 'r') as file:

sentences = file.readlines()

for sentence in sentences:

sentence = sentence.strip()

parsed\_sentences.append(parse\_sentence(sentence))

# In[ ]:

def get\_vocab(X\_train, Y\_train):

'''

Function for building the vocabulary from the training set of

words and tags.

'''

vocabulary2id = dict()

tag2id = dict()

vocabulary2id['PAD'] = 0

vocabulary2id['UNK'] = 1

for sent in X\_train:

for word in sent:

if word not in vocabulary2id.keys():

vocabulary2id[word] = len(vocabulary2id)

tag2id['PAD'] = 0

for sent in Y\_train:

for tag in sent:

if tag not in tag2id.keys():

tag2id[tag] = len(tag2id)

return vocabulary2id, tag2id

def get\_word\_tag\_counts(X\_train, Y\_train, vocabulary2id, tag2id):

'''

Function for calculating the counts pertaining to the

individual word tags.

'''

wordcount = defaultdict(int)

tagcount = defaultdict(int)

tagpaircount = defaultdict(int)

tagtriplecount = defaultdict(int)

for sent in X\_train:

for word in sent:

wordcount[word] += 1

for sent in Y\_train:

for tag in sent:

tagcount[tag] += 1

for sent in Y\_train:

for i in range(len(sent) - 1):

tagpaircount[sent[i], sent[i + 1]] += 1

for sent in Y\_train:

for i in range(len(sent) - 2):

tagtriplecount[sent[i], sent[i + 1], sent[i + 2]] += 1

return wordcount, tagcount, tagpaircount, tagtriplecount

# In[7]:

#parsed\_sentences[:5]

# In[ ]:

#defining model architecture

def build\_model():

model = Sequential()

model.add(Masking(mask\_value=float(vocabulary2id['UNK']),input\_shape=(padlen,)))

model.add(Embedding(len(vocabulary2id), 100))

model.add(Bidirectional(LSTM(int((128+256)/2), return\_sequences=True)))

model.add(TimeDistributed(Dense(len(tag2id))))

model.add(Activation('softmax'))

model.compile(loss='categorical\_crossentropy',

optimizer=Adam(0.001),

metrics=['accuracy'])

model.summary()

return model

# In[ ]:

#one hot representation of data

def id2onehot(Y, numtags):

out = []

for s in Y:

categories = []

for item in s:

categories.append(np.zeros(numtags))

categories[-1][item] = 1.0

out.append(categories)

return np.array(out)

# In[10]:

# Build the test and training sets of sentences in cross validation setup.

kf = KFold(n\_splits = 3, shuffle = False)

parsed\_sentences = np.asarray(parsed\_sentences)

scores = []

scores1 = []

y\_pred\_idx = []

y\_pred\_idx1 = []

y\_test\_idx = []

y\_test\_idx1 = []

preds\_all\_folds = []

golds\_all\_folds = []

for fold\_num, (train\_index, test\_index) in enumerate(kf.split(parsed\_sentences)):

train\_data = parsed\_sentences[train\_index]

test\_data = parsed\_sentences[test\_index]

X\_train = [a[0] for a in train\_data]

Y\_train = [a[1] for a in train\_data]

X\_test = [a[0] for a in test\_data]

Y\_test = [a[1] for a in test\_data]

# Build the vocabulary and word counts.

vocabulary2id, tag2id = get\_vocab(X\_train, Y\_train)

padlen = max(len(i) for i in X\_train)

def pad(sentence, padid=vocabulary2id['PAD']):

out = sentence[:padlen]

padding = [padid for \_ in range(padlen - len(out))]

return out + padding

X\_train\_ids = np.asarray([pad([vocabulary2id[word] if word in vocabulary2id.keys() else vocabulary2id['UNK'] for word in sent]) for sent in X\_train])

X\_test\_ids = np.array([pad([vocabulary2id[word] if word in vocabulary2id.keys() else vocabulary2id['UNK'] for word in sent]) for sent in X\_test])

Y\_train\_ids = np.asarray([pad([tag2id[word] if word in tag2id.keys() else tag2id['UNK'] for word in sent], tag2id['PAD']) for sent in Y\_train])

Y\_test\_ids = np.asarray([pad([tag2id[word] if word in tag2id.keys() else tag2id['UNK'] for word in sent], tag2id['PAD']) for sent in Y\_test])

Y\_train\_onehot = id2onehot(Y\_train\_ids, len(tag2id))

Y\_test\_onehot = id2onehot(Y\_test\_ids, len(tag2id))

model = build\_model()

model.fit(X\_train\_ids, Y\_train\_onehot, batch\_size=128, epochs=5, validation\_split=0.2)

predictions = model.predict(X\_test\_ids)

# test\_accuracy = np.sum((Y\_test\_ids == np.argmax(predictions, axis=-1)) \* (Y\_test\_ids != 0)) / np.sum((Y\_test\_ids != 0))

# print('Fold {} test\_accuracy: {}'.format(fold\_num + 1, test\_accuracy))

predictions\_argmax = np.argmax(predictions, axis=-1)

y\_pred\_nopad = []

y\_true\_nopad = []

for i in range(len(Y\_test\_ids)):

for j in range(len(Y\_test\_ids[i])):

if Y\_test\_ids[i][j] != 0 and predictions\_argmax[i][j] != 0:

y\_true\_nopad.append(Y\_test\_ids[i][j])

if predictions\_argmax[i][j] == 0:

y\_pred\_nopad.append(1)

else:

y\_pred\_nopad.append(predictions\_argmax[i][j])

preds\_all\_folds.append(y\_pred\_nopad)

golds\_all\_folds.append(y\_true\_nopad)

y\_pred\_nopad = np.asarray(y\_pred\_nopad)

y\_true\_nopad = np.asarray(y\_true\_nopad)

test\_accuracy = (y\_pred\_nopad == y\_true\_nopad).mean()

print('Fold {} test\_accuracy: {}'.format(fold\_num + 1, test\_accuracy))

prec, rec, fscore, \_ = precision\_recall\_fscore\_support(y\_true\_nopad, y\_pred\_nopad, average = 'weighted')

print('Fold {} Precision: {} Recall: {} F1-Score: {}'.format(fold\_num + 1, prec, rec, fscore))

# In[11]:

print("---Averaged Results over all the epochs---")

test\_accuracy = (np.asarray(preds\_all\_folds[0]) == np.asarray(golds\_all\_folds[0])).mean()

print('Average K-Fold Test Accuracy: {}'.format(test\_accuracy))

prec, rec, fscore, \_ = precision\_recall\_fscore\_support(preds\_all\_folds[0], golds\_all\_folds[0], average = 'weighted')

print('Average K-Fold Precision: {} Recall: {} F1-Score: {}'.format(prec, rec, fscore))

# In[12]:

id2tag = {v: k for k, v in tag2id.items()}

print(classification\_report([id2tag[i] for i in golds\_all\_folds[0]], [id2tag[i] for i in preds\_all\_folds[0]]))

# In[ ]:

def cm\_analysis(y\_true, y\_pred, labels, ymap=None, figsize=(10,10)):

"""

Generate matrix plot of confusion matrix with pretty annotations.

The plot image is saved to disk.

args:

y\_true: true label of the data, with shape (nsamples,)

y\_pred: prediction of the data, with shape (nsamples,)

filename: filename of figure file to save

labels: string array, name the order of class labels in the confusion matrix.

use `clf.classes\_` if using scikit-learn models.

with shape (nclass,).

ymap: dict: any -> string, length == nclass.

if not None, map the labels & ys to more understandable strings.

Caution: original y\_true, y\_pred and labels must align.

figsize: the size of the figure plotted.

"""

if ymap is not None:

y\_pred = [ymap[yi] for yi in y\_pred]

y\_true = [ymap[yi] for yi in y\_true]

labels = [ymap[yi] for yi in labels]

cm = confusion\_matrix(y\_true, y\_pred, labels=labels)

cm\_sum = np.sum(cm, axis=1, keepdims=True)

cm\_perc = cm / cm\_sum.astype(float) \* 100

annot = np.empty\_like(cm).astype(str)

nrows, ncols = cm.shape

for i in range(nrows):

for j in range(ncols):

c = cm[i, j]

p = cm\_perc[i, j]

if i == j:

s = cm\_sum[i]

annot[i, j] = '%.1f%%\n%d/%d' % (p, c, s)

elif c == 0:

annot[i, j] = ''

else:

annot[i, j] = '%.1f%%\n%d' % (p, c)

cm = pd.DataFrame(cm, index=labels, columns=labels)

cm.index.name = 'Actual'

cm.columns.name = 'Predicted'

fig, ax = plt.subplots(figsize=figsize)

sns.heatmap(cm, annot=annot, fmt='', ax=ax)

plt.show()

# In[17]:

cm\_analysis([id2tag[i] for i in golds\_all\_folds[0]], [id2tag[i] for i in preds\_all\_folds[0]], sorted(tag2id.keys(), key=lambda k: tag2id[k])[1:], ymap=None, figsize=(14, 14))

# In[18]:

print("Tag-wise counts and accuracies\n--\n")

print("Tag\tGold Tag Count\tTotal predicted correctly\tTotal predicted incorrectly\tAccuracy")

gold\_counts = Counter([id2tag[i] for i in golds\_all\_folds[0]])

pred\_counts\_correct = Counter([id2tag[t] for i, t in enumerate(preds\_all\_folds[0]) if t == golds\_all\_folds[0][i]])

pred\_counts\_incorrect = Counter([id2tag[t] for i, t in enumerate(preds\_all\_folds[0]) if t != golds\_all\_folds[0][i]])

for tag in gold\_counts.keys():

print(f"{tag}\t{gold\_counts[tag]}\t\t{pred\_counts\_correct[tag]}\t\t\t\t{pred\_counts\_incorrect[tag]}\t\t\t\t{pred\_counts\_correct[tag] / gold\_counts[tag]}")