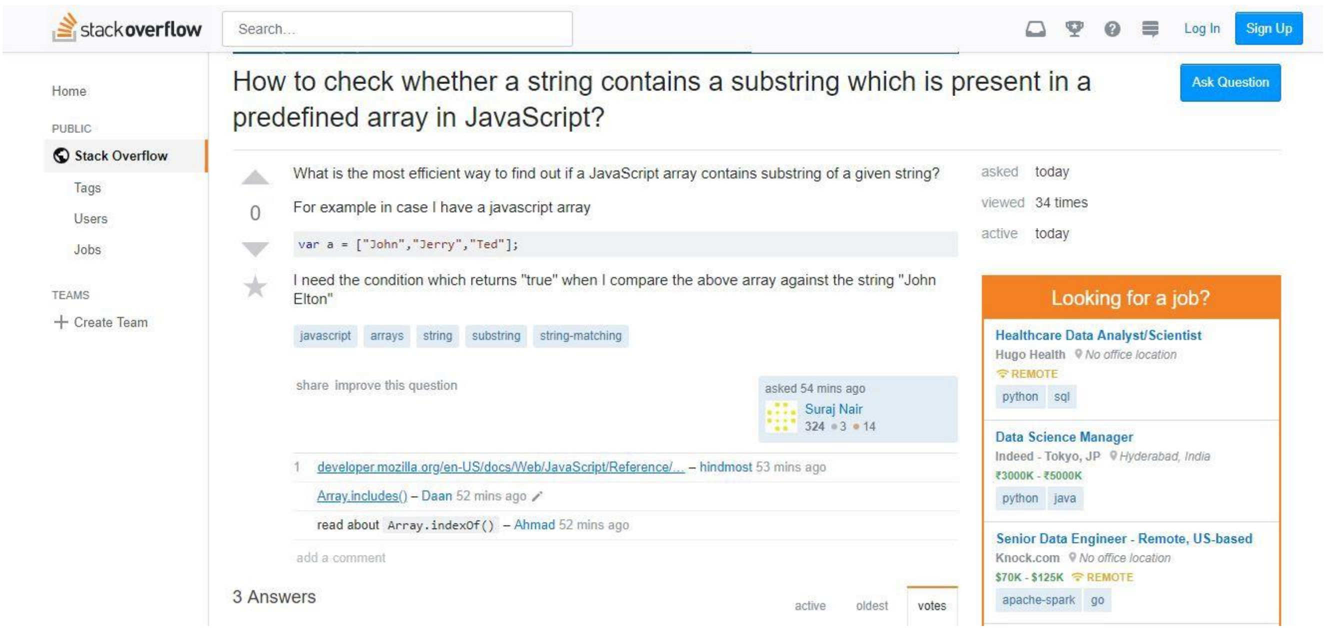
Assignment 3.1 – Stemming

Problem Statement: Predict tags on Stack Overflow with linear models

Theory:

One of the most common tasks of NLP is to automatically predict the topic of a question. In this assignment, we'll start from preprocessing Questions and tags of Stack Overflow and then we will build a simple model to predict the tag of a Stack Overflow question.

A question in Stack Overflow contains three segments Title, Description and Tags. By using the text in the title and description we should suggest the tags related to the subject of the question automatically. These tags are extremely important for the proper working of Stack Overflow.



How to check whether a string contains a substring which is present in a predefined array in JavaScript?

What is the most efficient way to find out if a JavaScript array contains substring of a given string?

For example in case I have a javascript array

```
var a = ["John","Jerry","Ted"];
```

I need the condition which returns "true" when I compare the above array against the string "John Elton"

javascript arrays string substring string-matching

asked 54 mins ago
Suraj Nair
324 ●3 ●14

1 developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/... – hindmost 53 mins ago

Array.includes() – Daan 52 mins ago

read about `Array.indexOf()` – Ahmad 52 mins ago

3 Answers

asked today
viewed 34 times
active today

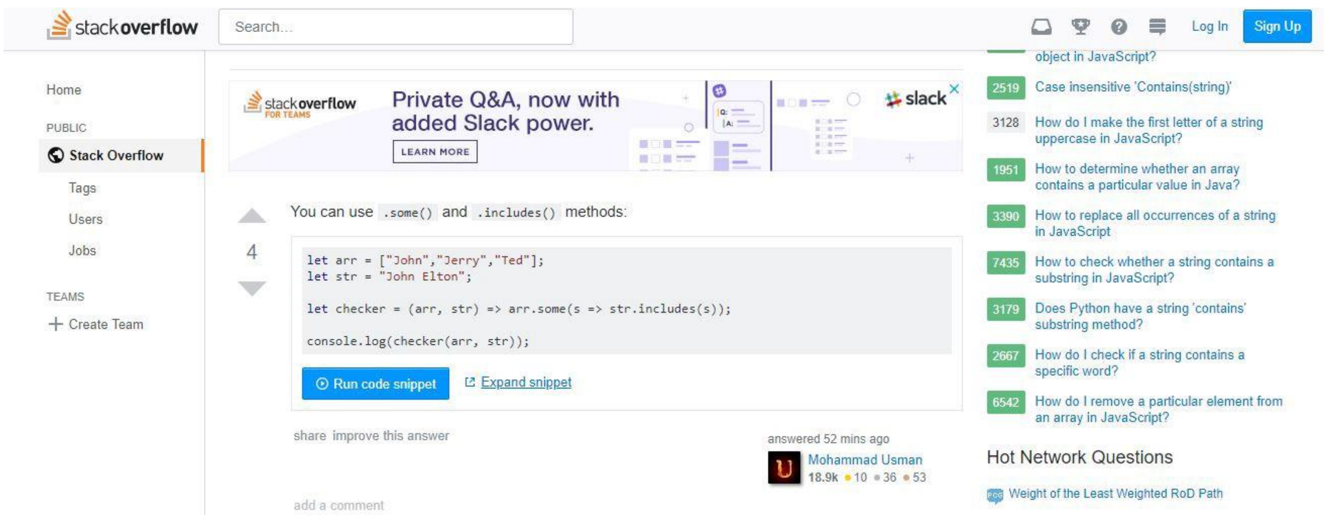
Looking for a job?

Healthcare Data Analyst/Scientist
Hugo Health No office location
REMOTE
python sql

Data Science Manager
Indeed - Tokyo, JP Hyderabad, India
₹3000K - ₹5000K
python java

Senior Data Engineer - Remote, US-based
Knock.com No office location
$70K - $125K REMOTE
apache-spark go

In above example a question was asked on Java Script. The user has given two lines of description and five tags.



You can use `.some()` and `.includes()` methods:

```
let arr = ["John","Jerry","Ted"];
let str = "John Elton";

let checker = (arr, str) => arr.some(s => str.includes(s));

console.log(checker(arr, str));
```

answered 52 mins ago
Mohammad Usman
18.9k ● 10 ● 36 ● 53

object in JavaScript?
2519 Case insensitive 'Contains(string)'
3128 How do I make the first letter of a string uppercase in JavaScript?
1951 How to determine whether an array contains a particular value in Java?
3390 How to replace all occurrences of a string in JavaScript
7435 How to check whether a string contains a substring in JavaScript?
3179 Does Python have a string 'contains' substring method?
2667 How do I check if a string contains a specific word?
6542 How do I remove a particular element from an array in JavaScript?

Hot Network Questions
Weight of the Least Weighted RoD Path

Stack overflow detects that the user who has answered the question has already done so to similar questions related to Java Script, Strings, Arrays etc in the past and recognizes that he is an expert in the subject. People can provide the tags related to the question on their own or Stack Overflow can predict the tags using the text in title and description. This is extremely business critical. The more accurately Stack Overflow can predict these tags the better it can create an Ecosystem to send the right question to the right set of people.

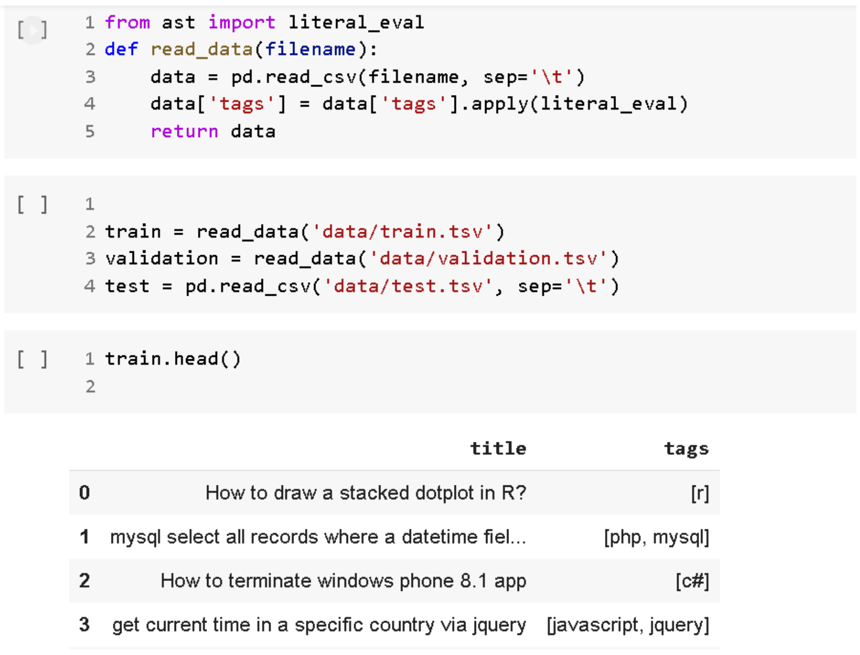
Business Objectives & Constraints

1. Predict as many tags as possible with high precision and recall.
2. Incorrect tags could impact customer experience on Stack Overflow
3. No Strict Latency Constraints

Machine Learning Procedure

1. **Obtain the Dataset**

We found a dataset online in TSV form, which we cleaned up by removing unneeded parameters with the help of Microsoft Excel to generate our current dataset The dataset was loaded with the help of Pandas



```python
from ast import literal_eval
def read_data(filename):
    data = pd.read_csv(filename, sep='\t')
    data['tags'] = data['tags'].apply(literal_eval)
    return data
```

```python

train = read_data('data/train.tsv')
validation = read_data('data/validation.tsv')
test = pd.read_csv('data/test.tsv', sep='\t')
```

```python
train.head()

```

	title	tags
0	How to draw a stacked dotplot in R?	[r]
1	mysql select all records where a datetime fiel...	[php, mysql]
2	How to terminate windows phone 8.1 app	[c#]
3	get current time in a specific country via jquery	[javascript, jquery]

2. Perform Cleanup

Step 1: CONVERT THE SENTENCE TO IT'S SIMPLEST FORM

As our algorithm, which does not rely on order or context, removing common words or symbols will not cause any major issue

Step 2: STOPWORD REMOVAL

Stopwords refer to the most common word which while provide context, play no role in a project like ours which does not rely on the context they can provide

Step 3: REMOVE UNUSED SYMBOLS

{, }, [,] etc do not play any role in our project and hence can be removed safely

Step 4: CHANGE CASE

As upper and lower case would lead to different words, even if the meaning is the same, it is best to change to lower case so as to increase the size of the dataset instead of removing the words.

```python
pd.DataFrame({'prepared': X_train_prepare[:10], 'original': X_train[:10]})
```

```
                                      prepared                                         original
0                          draw stacked dotplot r                  How to draw a stacked dotplot in R?
1  mysql select records datetime field less speci...   mysql select all records where a datetime fiel...
2                terminate windows phone 81 app           How to terminate windows phone 8.1 app
3       get current time specific country via jquery     get current time in a specific country via jquery
4                  configuring tomcat use ssl                      Configuring Tomcat to Use SSL
5  awesome nested set plugin add new children tre...  Awesome nested set plugin - how to add new chi...
6           create map json response ruby rails 3   How to create map from JSON response in Ruby o...
7                     rspec test method called                      rspec test if method is called
8           springboot catalina lifecycle exception          SpringBoot Catalina LifeCycle Exception
9       import data excel mysql database using php     How to import data from excel to mysql databas...
```

3. Perform TF-IDF

TF-IDF is used to find the relevance of a word within the document

TERM FREQUENCY

The frequency of a given word in a document. The weight of a word in a Document is simply proportional to it's Term Frequency

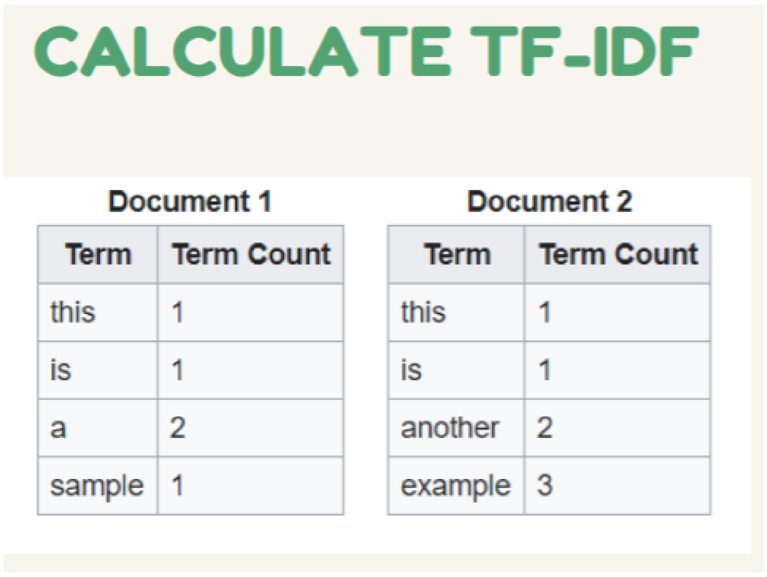
INVERSE DOCUMENT FREQUENCY

It is a measure of how much information the word provides. As such, we check how rare or common it is within the document. It is the logarithmically scaled inverse fraction of the documents that contain the word (obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient).

TF-IDF

Reflects how important a word is to a document in a collection or corpus

Multiply TF * IDF



CALCULATE TF-IDF

Document 1

Term	Term Count
this	1
is	1
a	2
sample	1

Document 2

Term	Term Count
this	1
is	1
another	2
example	3

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()

X_train_tfidf = tfidf_vectorizer.fit_transform(X_train_prepare)
X_val_tfidf = tfidf_vectorizer.transform(X_val_prepare)
X_test_tfidf = tfidf_vectorizer.transform(X_test_prepare)
tfidf_vocabulary = tfidf_vectorizer.vocabulary_

```

```
print(X_train_tfidf[1])
```

```
  (0, 29588)	0.24540274618798358
  (0, 25494)	0.4064986946612677
  (0, 15353)	0.4454941760104249
  (0, 10048)	0.32287114158489333
  (0, 7133)	0.3649429374739481
  (0, 22418)	0.40323640938975913
  (0, 24149)	0.30980412500230314
  (0, 17676)	0.28295569111866237
```

4. Convert to Multi label Binarizer

   Helps make it easier for Linear Model to comprehend the topic

```
from sklearn.preprocessing import MultiLabelBinarizer

y_tags = list(set(np.concatenate(y_train)))

mlb = MultiLabelBinarizer(classes=sorted(set(y_tags)))
y_train_mlb = mlb.fit_transform(y_train)
y_val_mlb = mlb.fit_transform(y_val)
```

```
print(y_train_mlb[1])
print(mlb.inverse_transform(y_train_mlb)[1])
print(X_train[1])
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
('mysql', 'php')
mysql select all records where a datetime field is less than a specified value
```

5. Train and Test our Linear Model

   Check what went right and what went wrong

   1. MULTINOMIAL LOGISTIC REGRESSION
   2. ONE vs REST

      Helps perform Multiclass Classification Compare one value (C++) with all of the other labels

```
Title:  Why odbc_exec always fail?
True labels:    php,sql
Predicted labels:


Title:  Access a base classes variable from within a child class
True labels:    javascript
Predicted labels:       class


Title:  Content-Type "application/json" not required in rails
True labels:    ruby,ruby-on-rails
Predicted labels:       json,ruby-on-rails


Title:  Sessions in Sinatra: Used to Pass Variable
True labels:    ruby,session
Predicted labels:


Title:  Getting error - type "json" does not exist - in Postgresql during rake db migrate
True labels:    json,ruby,ruby-on-rails
Predicted labels:       ruby-on-rails
```

Code And Output:

Link to the colab notebook:
https://drive.google.com/file/d/1A-KU0waboYurX4LDsgIi8nj7LimLIG-L/view?usp=sharing

```
# -*- coding: utf-8 -*-
"""so.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1A-KU0waboYurX4LDsgIi8nj7LimLIG-L
"""

from google.colab import drive
drive.mount('/content/drive')

!pip3 install nltk pandas numpy sklearn matplotlib

cd /content/drive/My Drive/StackOverflow

import pandas as pd
import numpy as np

from ast import literal_eval
def read_data(filename):
    data = pd.read_csv(filename, sep='\t')
    data['tags'] = data['tags'].apply(literal_eval)
    return data

train = read_data('data/train.tsv')
validation = read_data('data/validation.tsv')
test = pd.read_csv('data/test.tsv', sep='\t')
```

```
train.head()

test.head()

validation.head()

X_train, y_train = train['title'].values, train['tags'].values
X_val, y_val = validation['title'].values, validation['tags'].values
X_test = test['title'].values

import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords

stopwords_english = set(stopwords.words('english'))
list(stopwords_english)[:10]

import re
REPLACE_BY_SPACE_RE = re.compile('[/(){}\[\]\|@,;]')
# Any symbols other than these are removed
BAD_SYMBOLS_RE = re.compile('[^0-9a-z #+_]')
STOPWORDS = set(stopwords.words('english'))

def text_prepare(text):
    """
        text: a string

        return: modified initial string
    """
    text = text.lower()
    text = re.sub(REPLACE_BY_SPACE_RE, " ", text)
    text = " ".join([word for word in text.split(" ") if word not in stopwords_englis
h])
    text = re.sub(BAD_SYMBOLS_RE, "", text)
    text = " ".join([word for word in text.split(" ") if len(word) != 0])
    return text

text_prepare("SQL Server - any equivalent of Excel's CHOOSE function?")

X_train_prepare = [text_prepare(question) for question in X_train]
X_test_prepare = [text_prepare(question) for question in X_test]
X_val_prepare = [text_prepare(question) for question in X_val]

pd.DataFrame({'prepared': X_train_prepare[:10], 'original': X_train[:10]})

from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()

X_train_tfidf = tfidf_vectorizer.fit_transform(X_train_prepare)
X_val_tfidf = tfidf_vectorizer.transform(X_val_prepare)
X_test_tfidf = tfidf_vectorizer.transform(X_test_prepare)
```

```
tfidf_vocabulary = tfidf_vectorizer.vocabulary_

print(X_train_tfidf[1])

from sklearn.preprocessing import MultiLabelBinarizer

y_tags = list(set(np.concatenate(y_train)))

mlb = MultiLabelBinarizer(classes=sorted(set(y_tags)))
y_train_mlb = mlb.fit_transform(y_train)
y_val_mlb = mlb.fit_transform(y_val)

print(y_train_mlb[1])
print(mlb.inverse_transform(y_train_mlb)[1])
print(X_train[1])

from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression
classifier_tfidf  = LogisticRegression(solver='liblinear')
classifier_tfidf = OneVsRestClassifier(classifier_tfidf)
classifier_tfidf.fit(X_train_tfidf, y_train_mlb)

y_val_predicted_labels_tfidf = classifier_tfidf.predict(X_val_tfidf)
y_val_predicted_scores_tfidf = classifier_tfidf.decision_function(X_val_tfidf)

y_val_pred_inversed = mlb.inverse_transform(y_val_predicted_labels_tfidf)
y_val_inversed = mlb.inverse_transform(y_val_mlb)
for (question, label, pred) in zip(X_val[0:5], y_val_inversed, y_val_pred_inversed):
    print('Title:\t{}\nTrue labels:\t{}\nPredicted labels:\t{}\n\n'.format(
        question,
        ','.join(label),
        ','.join(pred)
    ))

from sklearn.metrics import roc_auc_score, f1_score
f1_score = f1_score(y_val_mlb, y_val_predicted_labels_tfidf, average='weighted') * 10
0
print("F1 Score" ,f1_score, "%")

# Commented out IPython magic to ensure Python compatibility.
import matplotlib.pyplot as plt
# %matplotlib inline

from collections import Counter
y_train_all_vals = np.concatenate(y_train)
y_train_freq = Counter(y_train_all_vals)
avg = list(y_train_freq.items())[int(len(y_train_freq) * 0.5)]
avg = [avg for _ in range(0,6)]
pd.DataFrame({"most_common": y_train_freq.most_common(6),
"least_common": y_train_freq.most_common()[-6:], "average_val": avg})
```

```
plt.hist(y_train_freq.values())

mlb.inverse_transform(classifier_tfidf.predict(tfidf_vectorizer.transform(["visual c+
+"])))
```

Conclusion:

Using data preproceesing and linear algorithms, we were successfully able to distinguish tags on stack overflow questions.

Assignment 3.1 – Stemming

Problem Statement: Recognize named entities on Twitter with LSTMs

Theory:

NER is a common task in natural language processing systems. It serves for extraction such entities from the text as persons, organizations, locations, etc. In this task we will experiment to recognize named entities from Twitter.

Let’s say we want to extract

- the person names

- the company names
- the location names
- the music artist names
- the tv show names

For example, we want to extract persons' and organizations' names from the text. Then for the input text:

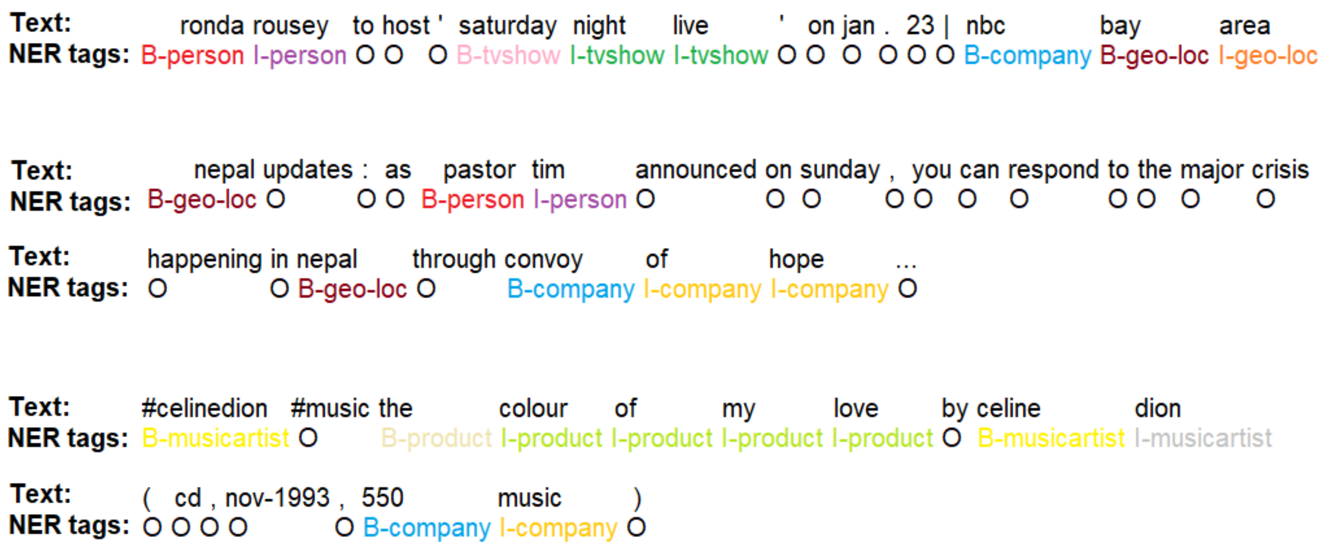
```
Ian Goodfellow works for Google Brain
```

a NER model needs to provide the following sequence of tags:

```
B-PER I-PER    O     O   B-ORG  I-ORG
```

Where B- and I- prefixes stand for the beginning and inside of the entity, while O stands for out of tag or no tag. Markup with the prefix scheme is called BIO markup. This markup is introduced for distinguishing of consequent entities with similar types.

More examples are shown below in the diagram


Text: ronda rousey to host ' saturday night live ' on jan . 23 | nbc bay area
NER tags: B-person I-person O O O B-tvshow I-tvshow I-tvshow O O O O O O B-company B-geo-loc I-geo-loc

Text: nepal updates : as pastor tim announced on sunday , you can respond to the major crisis
NER tags: B-geo-loc O O O B-person I-person O O O O O O O O O O O

Text: happening in nepal through convoy of hope ...
NER tags: O O B-geo-loc O B-company I-company I-company O

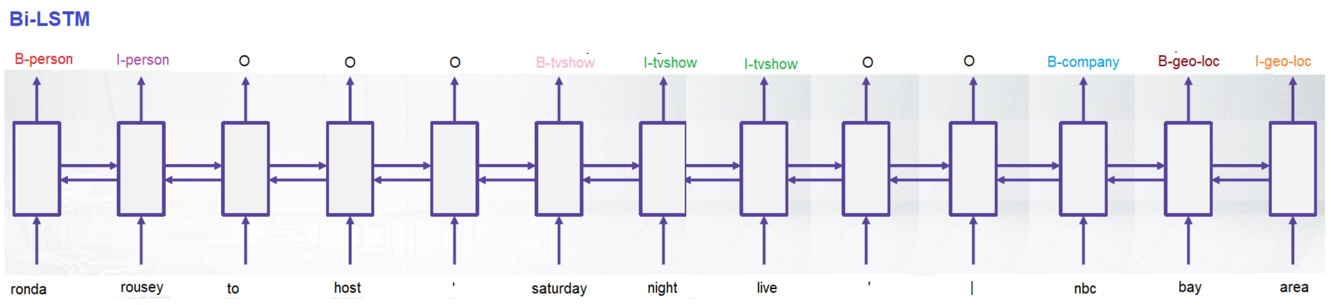
Text: #celinedion #music the colour of my love by celine dion
NER tags: B-musicartist O B-product I-product I-product I-product I-product O B-musicartist I-musicartist

Text: (cd , nov-1993 , 550 music)
NER tags: O O O O O B-company I-company O

A solution of the task will be based on neural networks, particularly, on Bi-Directional Long Short-Term Memory Networks (Bi-LSTMs).

Bi-LSTM

- Provides a universal approach for sequence tagging
- Several layers can be stacked + linear layers can be added on top
- Is trained by cross-entropy loss coming from each position



Code And Output:

Link to the colab notebook:
https://drive.google.com/file/d/119E7Y6OJGRaSf2dTsMmKrCtyj2HHn4QU/view?usp=sharing

```
# -*- coding: utf-8 -*-
"""assignment3-Twitter-NER.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/119E7Y6OJGRaSf2dTsMmKrCtyj2HHn4QU
"""

from google.colab import drive
drive.mount('/content/drive')

cd /content/drive/My Drive/LSTM

"""# Recognize named entities on Twitter with LSTMs

In this assignment, we will use a recurrent neural network to solve Named Entity Reco
gnition (NER) problem. NER is a common task in natural language processing systems. I
t serves for extraction such entities from the text as persons, organizations, locati
ons, etc. In this task we will experiment to recognize named entities from Twitter.

Let’s say we want to extract

- the person names
- the company names
- the location names
- the music artist names
- the tv show names

For example, we want to extract persons' and organizations' names from the text. Than
 for the input text:

    Ian Goodfellow works for Google Brain

a NER model needs to provide the following sequence of tags:

    B-PER I-PER    O      O    B-ORG   I-ORG
```

```
Where *B-* and *I-
* prefixes stand for the beginning and inside of the entity, while *O* stands for out
 of tag or no tag. Markup with the prefix scheme is called *BIO markup*. This markup
is introduced for distinguishing of consequent entities with similar types.

More examples are shown below in the diagram
![picture](https://sandipanweb.files.wordpress.com/2020/08/ner-ex.png)

A solution of the task will be based on neural networks, particularly, on Bi-
Directional Long Short-Term Memory Networks (Bi-LSTMs).

### Bi-LSTM

- Provides a universal approach for sequence tagging

- Several layers can be stacked + linear layers can be added on top

- Is trained by cross-entropy loss coming from each position

![picture](https://sandipanweb.files.wordpress.com/2020/08/bilstm-1.png)

### Libraries

For this task we will need the following libraries:
 - [Tensorflow](https://www.tensorflow.org) — an open-
source software library for Machine Intelligence.

In this assignment, we use Tensorflow 1.15.0. we can install it with pip:

    !pip install tensorflow==1.15.0

 - [Numpy](http://www.numpy.org) — a package for scientific computing.

If we have never worked with Tensorflow, we would probably need to read some tutorial
s during our work on this assignment, e.g. [this one](https://www.tensorflow.org/tuto
rials/recurrent) could be a good starting point.

### Data

The following cell will download all data required for this assignment into the folde
r `lstm/data`.
"""

import sys
sys.path.append("..")

"""### Load the Twitter Named Entity Recognition corpus
```

```
We will work with a corpus, which contains tweets with NE tags. Every line of a file
contains a pair of a token (word/punctuation symbol) and a tag, separated by a whites
pace. Different tweets are separated by an empty line.

The function *read_data* reads a corpus from the *file_path* and returns two lists: o
ne with tokens and one with the corresponding tags. We need to complete this function
 by adding a code, which will replace a user's nickname to `<USR>` token and any URL
to `<URL>` token. We could think that a URL and a nickname are just strings which sta
rt with *http://* or *https://* in case of URLs and a *@* symbol for nicknames.
"""

def read_data(file_path):
    tokens = []
    tags = []

    tweet_tokens = []
    tweet_tags = []
    for line in open(file_path, encoding='utf-8'):
        line = line.strip()
        if not line:
            if tweet_tokens:
                tokens.append(tweet_tokens)
                tags.append(tweet_tags)
            tweet_tokens = []
            tweet_tags = []
        else:
            token, tag = line.split()
            # Replace all urls with <URL> token
            # Replace all users with <USR> token

            if token.startswith('@'):
                token = '<USR>'
            elif token.startswith('http://') or token.startswith('https://'):
                token = '<URL>'

            tweet_tokens.append(token)
            tweet_tags.append(tag)

    return tokens, tags

"""And now we can load three separate parts of the dataset:
 - *train* data for training the model;
 - *validation* data for evaluation and hyperparameters tuning;
 - *test* data for final evaluation of the model.
"""

train_tokens, train_tags = read_data('data/train.txt')
validation_tokens, validation_tags = read_data('data/validation.txt')
test_tokens, test_tags = read_data('data/test.txt')
```

```
"""We should always understand what kind of data we deal with. For this purpose, we c
an print the data running the following cell:"""

for i in range(3):
    for token, tag in zip(train_tokens[i], train_tags[i]):
        print('%s\t%s' % (token, tag))
    print()

"""### Prepare dictionaries

To train a neural network, we will use two mappings:
- {token}$\to${token id}: address the row in embeddings matrix for the current token;
- {tag}$\to${tag id}: one-
hot ground truth probability distribution vectors for computing the loss at the outpu
t of the network.

Now we need to implement the function *build_dict* which will return {token or tag}$\
to${index} and vice versa.
"""

from collections import defaultdict

def build_dict(tokens_or_tags, special_tokens):
    """
        tokens_or_tags: a list of lists of tokens or tags
        special_tokens: some special tokens
    """
    # Create a dictionary with default value 0
    tok2idx = defaultdict(lambda: 0)
    idx2tok = []

    # Create mappings from tokens (or tags) to indices and vice versa.
    # At first, add special tokens (or tags) to the dictionaries.
    # The first special token must have index 0.

    # Mapping tok2idx should contain each token or tag only once.
    # To do so, you should:
    # 1. extract unique tokens/tags from the tokens_or_tags variable, which is not
    #    occur in special_tokens (because they could have non-empty intersection)
    # 2. index them (for example, you can add them into the list idx2tok
    # 3. for each token/tag save the index into tok2idx).

    for i, token in enumerate(special_tokens):
        tok2idx[token] = i
        idx2tok.append(token)

    nextIndex = len(special_tokens)
    for tokens in tokens_or_tags:
        for token in tokens:
            if token not in tok2idx:
                tok2idx[token] = nextIndex
                idx2tok.append(token)
                nextIndex += 1

    return tok2idx, idx2tok

"""After implementing the function *build_dict* we  make dictionaries for tokens and 
tags. Special tokens in our case will be:
 - `<UNK>` token for out of vocabulary tokens;
 - `<PAD>` token for padding sentence to the same length when we create batches of se
ntences.
"""

special_tokens = ['<UNK>', '<PAD>']
special_tags = ['O']

# Create dictionaries
token2idx, idx2token = build_dict(train_tokens + validation_tokens, special_tokens)
tag2idx, idx2tag = build_dict(train_tags, special_tags)

"""The next additional functions will help to create the mapping between tokens and i
ds for a sentence. """

def words2idxs(tokens_list):
    return [token2idx[word] for word in tokens_list]

def tags2idxs(tags_list):
    return [tag2idx[tag] for tag in tags_list]

def idxs2words(idxs):
    return [idx2token[idx] for idx in idxs]

def idxs2tags(idxs):
    return [idx2tag[idx] for idx in idxs]

"""### Generate batches

Neural Networks are usually trained with batches. It means that weight updates of the
 network are based on several sequences at every single time. The tricky part is that
 all sequences within a batch need to have the same length. So we will pad them with 
a special `<PAD>` token. It is also a good practice to provide RNN with sequence leng
ths, so it can skip computations for padding parts. We provide the batching function 
*batches_generator* readily available for you to save time.
"""

def batches_generator(batch_size, tokens, tags,
                      shuffle=True, allow_smaller_last_batch=True):
    """Generates padded batches of tokens and tags."""

    n_samples = len(tokens)
    if shuffle:
        order = np.random.permutation(n_samples)
    else:
        order = np.arange(n_samples)

    n_batches = n_samples // batch_size
    if allow_smaller_last_batch and n_samples % batch_size:
        n_batches += 1

    for k in range(n_batches):
        batch_start = k * batch_size
        batch_end = min((k + 1) * batch_size, n_samples)
        current_batch_size = batch_end - batch_start
        x_list = []
        y_list = []
        max_len_token = 0
        for idx in order[batch_start: batch_end]:
            x_list.append(words2idxs(tokens[idx]))
            y_list.append(tags2idxs(tags[idx]))
            max_len_token = max(max_len_token, len(tags[idx]))

        # Fill in the data into numpy nd-arrays filled with padding indices.
        x = np.ones([current_batch_size, max_len_token], dtype=np.int32) * token2idx[
'<PAD>']
        y = np.ones([current_batch_size, max_len_token], dtype=np.int32) * tag2idx['O
']
        lengths = np.zeros(current_batch_size, dtype=np.int32)
        for n in range(current_batch_size):
            utt_len = len(x_list[n])
            x[n, :utt_len] = x_list[n]
            lengths[n] = utt_len
            y[n, :utt_len] = y_list[n]
        yield x, y, lengths

"""## Build a recurrent neural network

This is the most important part of the assignment. Here we will specify the network a
rchitecture based on TensorFlow building blocks. It's fun and easy as a lego construc
tor! We will create an LSTM network which will produce probability distribution over
tags for each token in a sentence. To take into account both right and left contexts
of the token, we will use Bi-Directional LSTM (Bi-
LSTM). Dense layer will be used on top to perform tag classification.
"""

import tensorflow as tf
import numpy as np

class BiLSTMModel():
    pass

"""First, we need to create [placeholders](https://www.tensorflow.org/api_docs/python
/tf/compat/v1/placeholder) to specify what data we are going to feed into the network
 during the execution time.  For this task we will need the following placeholders:
```

```
 - *input_batch* — sequences of words (the shape equals to [batch_size, sequence_len]
);
 - *ground_truth_tags* — sequences of tags (the shape equals to [batch_size, sequence
_len]);
 - *lengths* — lengths of not padded sequences (the shape equals to [batch_size]);
 - *dropout_ph* — dropout keep probability; this placeholder has a predefined value 1
;
 - *learning_rate_ph* — learning rate; we need this placeholder because we want to ch
ange the value during training.

It could be noticed that we use *None* in the shapes in the declaration, which means
that data of any size can be feeded.

You need to complete the function *declare_placeholders*.
"""

def declare_placeholders(self):
    """Specifies placeholders for the model."""

    # Placeholders for input and ground truth output.
    self.input_batch = tf.compat.v1.placeholder(dtype=tf.int32, shape=[None, None], n
ame='input_batch')
    self.ground_truth_tags =  tf.compat.v1.placeholder(dtype=tf.int32, shape=[None, N
one], name='ground_truth_tags')

    # Placeholder for lengths of the sequences.
    self.lengths =  tf.compat.v1.placeholder(dtype=tf.int32, shape=[None], name='leng
ths')

    # Placeholder for a dropout keep probability. If we don't feed
    # a value for this placeholder, it will be equal to 1.0.
    self.dropout_ph =  tf.compat.v1.placeholder_with_default(tf.cast(1.0, tf.float32)
, shape=[])

    # Placeholder for a learning rate (tf.float32).
    self.learning_rate_ph =  tf.compat.v1.placeholder(dtype=tf.float32, shape=[])

BiLSTMModel.__declare_placeholders = classmethod(declare_placeholders)

"""Now, let us specify the layers of the neural network. First, we need to perform so
me preparatory steps:

- Create embeddings matrix with [tf.Variable](https://www.tensorflow.org/api_docs/pyt
hon/tf/Variable). Specify its name (*embeddings_matrix*), type  (*tf.float32*), and i
nitialize with random values.
- Create forward and backward LSTM cells. TensorFlow provides a number of RNN cells r
eady for you. We suggest that you use *LSTMCell*, but you can also experiment with ot
her types, e.g. GRU cells. [This](http://colah.github.io/posts/2015-08-Understanding-
LSTMs/) blogpost could be interesting if you want to learn more about the differences
.
```

```
- Wrap your cells with [DropoutWrapper](https://www.tensorflow.org/api_docs/python/tf
/contrib/rnn/DropoutWrapper). Dropout is an important regularization technique for ne
ural networks. Specify all keep probabilities using the dropout placeholder that we c
reated before.

After that, we build the computation graph that transforms an input_batch:

- [Look up](https://www.tensorflow.org/api_docs/python/tf/nn/embedding_lookup) embedd
ings for an *input_batch* in the prepared *embedding_matrix*.
- Pass the embeddings through [Bidirectional Dynamic RNN](https://www.tensorflow.org/
api_docs/python/tf/nn/bidirectional_dynamic_rnn) with the specified forward and backw
ard cells. Use the lengths placeholder here to avoid computations for padding tokens
inside the RNN.
- Create a dense layer on top. Its output will be used directly in loss function.

In case you need to debug something, the easiest way is to check that tensor shapes o
f each step match the expected ones.

"""

def build_layers(self, vocabulary_size, embedding_dim, n_hidden_rnn, n_tags):
    """Specifies bi-LSTM architecture and computes logits for inputs."""

    # Create embedding variable (tf.Variable) with dtype tf.float32
    initial_embedding_matrix = np.random.randn(vocabulary_size, embedding_dim) / np.s
qrt(embedding_dim)
    embedding_matrix_variable = tf.Variable(initial_embedding_matrix, dtype=tf.float3
2)

    # Create RNN cells (for example, tf.nn.rnn_cell.BasicLSTMCell) with n_hidden_rnn
number of units
    # and dropout (tf.nn.rnn_cell.DropoutWrapper), initializing all *_keep_prob with
dropout placeholder.
    forward_cell =  tf.compat.v1.nn.rnn_cell.DropoutWrapper( tf.compat.v1.nn.rnn_cell
.LSTMCell(n_hidden_rnn),
                                                 input_keep_prob=self.dropout_ph,
                                                 output_keep_prob=self.dropout_ph,
                                                 state_keep_prob=self.dropout_ph)
    backward_cell =  tf.compat.v1.nn.rnn_cell.DropoutWrapper( tf.compat.v1.nn.rnn_cel
l.LSTMCell(n_hidden_rnn),
                                                  input_keep_prob=self.dropout_ph,
                                                  output_keep_prob=self.dropout_ph,
                                                  state_keep_prob=self.dropout_ph)

    # Look up embeddings for self.input_batch (tf.nn.embedding_lookup).
    # Shape: [batch_size, sequence_len, embedding_dim].
    embeddings =  tf.nn.embedding_lookup(embedding_matrix_variable, self.input_batch)

    # Pass them through Bidirectional Dynamic RNN (tf.nn.bidirectional_dynamic_rnn).
    # Shape: [batch_size, sequence_len, 2 * n_hidden_rnn].
    # Also don't forget to initialize sequence_length as self.lengths and dtype as tf
.float32.
    (rnn_output_fw, rnn_output_bw), _ =  tf.compat.v1.nn.bidirectional_dynamic_rnn(ce
ll_fw=forward_cell,
                                                                                 cell_bw=backw
ard_cell,
                                                                                 inputs=embedd
ings,
                                                                                 sequence_leng
th=self.lengths,
                                                                                 dtype=tf.floa
t32)
    rnn_output = tf.concat([rnn_output_fw, rnn_output_bw], axis=2)

    # Dense layer on top.
    # Shape: [batch_size, sequence_len, n_tags].
    self.logits =  tf.compat.v1.layers.dense(rnn_output, n_tags, activation=None)

BiLSTMModel.__build_layers = classmethod(build_layers)

"""To compute the actual predictions of the neural network, we apply [softmax](https:
//www.tensorflow.org/api_docs/python/tf/nn/softmax) to the last layer and find the mo
st probable tags with [argmax](https://www.tensorflow.org/api_docs/python/tf/argmax).
"""

def compute_predictions(self):
    """Transforms logits to probabilities and finds the most probable tags."""

    # Create softmax (tf.nn.softmax) function
    softmax_output = tf.nn.softmax(self.logits)

    # Use argmax (tf.argmax) to get the most probable tags
    # Don't forget to set axis=-1
    # otherwise argmax will be calculated in a wrong way
    self.predictions = tf.argmax(softmax_output, axis=-1)

BiLSTMModel.__compute_predictions = classmethod(compute_predictions)

"""During training we do not need predictions of the network, but we need a loss func
tion. We will use [cross-entropy loss](http://ml-
cheatsheet.readthedocs.io/en/latest/loss_functions.html#cross-
entropy), efficiently implemented in TF as
[cross entropy with logits](https://www.tensorflow.org/api_docs/python/tf/nn/softmax_
cross_entropy_with_logits_v2). Note that it should be applied to logits of the model
(not to softmax probabilities!). Also note,  that we do not want to take into account
 loss terms coming from `<PAD>` tokens. So we need to mask them out, before computing
 [mean](https://www.tensorflow.org/api_docs/python/tf/reduce_mean).
"""

def compute_loss(self, n_tags, PAD_index):
    """Computes masked cross-entopy loss with logits."""
    # Create cross entropy function function (tf.nn.softmax_cross_entropy_with_logits
_v2)
    ground_truth_tags_one_hot = tf.one_hot(self.ground_truth_tags, n_tags)
    loss_tensor =   tf.compat.v1.nn.softmax_cross_entropy_with_logits_v2(labels=groun
d_truth_tags_one_hot, logits=self.logits)

    mask = tf.cast(tf.not_equal(self.input_batch, PAD_index), tf.float32)
    # Create loss function which doesn't operate with <PAD> tokens (tf.reduce_mean)
    # Be careful that the argument of tf.reduce_mean should be
    # multiplication of mask and loss_tensor.
    self.loss =  tf.reduce_mean(mask*loss_tensor)

BiLSTMModel.__compute_loss = classmethod(compute_loss)

"""The last thing to specify is how we want to optimize the loss.
We suggest that you use [Adam](https://www.tensorflow.org/api_docs/python/tf/train/Ad
amOptimizer) optimizer with a learning rate from the corresponding placeholder.
You will also need to apply clipping to eliminate exploding gradients. It can be easi
ly done with [clip_by_norm](https://www.tensorflow.org/api_docs/python/tf/clip_by_nor
m) function.
"""

def perform_optimization(self):
    """Specifies the optimizer and train_op for the model."""

    # Create an optimizer (tf.train.AdamOptimizer)
    self.optimizer =  tf.compat.v1.train.AdamOptimizer(learning_rate=self.learning_ra
te_ph)
    self.grads_and_vars = self.optimizer.compute_gradients(self.loss)

    # Gradient clipping (tf.clip_by_norm) for self.grads_and_vars
    # Pay attention that you need to apply this operation only for gradients
    # because self.grads_and_vars also contains variables.
    # list comprehension might be useful in this case.
    clip_norm = tf.cast(1.0, tf.float32)
    self.grads_and_vars = [(tf.clip_by_norm(grad, clip_norm), var) for grad, var in s
elf.grads_and_vars]

    self.train_op = self.optimizer.apply_gradients(self.grads_and_vars)

BiLSTMModel.__perform_optimization = classmethod(perform_optimization)

"""Finally have specified all the parts of your network. We may have noticed, that we
 didn't deal with any real data yet, so what you have written is just recipes on how
the network should function.
Now we will put them to the constructor of our Bi-
LSTM class to use it in the next section.
"""
```

```
def init_model(self, vocabulary_size, n_tags, embedding_dim, n_hidden_rnn, PAD_index)
:
    self.__declare_placeholders()
    self.__build_layers(vocabulary_size, embedding_dim, n_hidden_rnn, n_tags)
    self.__compute_predictions()
    self.__compute_loss(n_tags, PAD_index)
    self.__perform_optimization()

BiLSTMModel.__init__ = classmethod(init_model)

"""## Train the network and predict tags

[Session.run](https://www.tensorflow.org/api_docs/python/tf/Session#run) is a point w
hich initiates computations in the graph that we have defined. To train the network,
we need to compute *self.train_op*, which was declared in *perform_optimization*. To
predict tags, we just need to compute *self.predictions*. Anyway, we need to feed act
ual data through the placeholders that we defined before.
"""

def train_on_batch(self, session, x_batch, y_batch, lengths, learning_rate, dropout_k
eep_probability):
    feed_dict = {self.input_batch: x_batch,
                 self.ground_truth_tags: y_batch,
                 self.learning_rate_ph: learning_rate,
                 self.dropout_ph: dropout_keep_probability,
                 self.lengths: lengths}

    session.run(self.train_op, feed_dict=feed_dict)

BiLSTMModel.train_on_batch = classmethod(train_on_batch)

"""Implement the function *predict_for_batch* by initializing *feed_dict* with input
*x_batch* and *lengths* and running the *session* for *self.predictions*."""

def predict_for_batch(self, session, x_batch, lengths):
    ######################################
    ######### YOUR CODE HERE #############
    ######################################
    predictions = session.run(self.predictions, feed_dict={self.input_batch:x_batch,
self.lengths:lengths})

    return predictions

BiLSTMModel.predict_for_batch = classmethod(predict_for_batch)

"""We finished with necessary methods of our BiLSTMModel model and almost ready to st
art experimenting.

### Evaluation
To simplify the evaluation process we provide two functions for you:
 - *predict_tags*: uses a model to get predictions and transforms indices to tokens a
nd tags;
 - *eval_conll*: calculates precision, recall and F1 for the results.
"""

from evaluation import precision_recall_f1

def predict_tags(model, session, token_idxs_batch, lengths):
    """Performs predictions and transforms indices to tokens and tags."""

    tag_idxs_batch = model.predict_for_batch(session, token_idxs_batch, lengths)

    tags_batch, tokens_batch = [], []
    for tag_idxs, token_idxs in zip(tag_idxs_batch, token_idxs_batch):
        tags, tokens = [], []
        for tag_idx, token_idx in zip(tag_idxs, token_idxs):
            tags.append(idx2tag[tag_idx])
            tokens.append(idx2token[token_idx])
        tags_batch.append(tags)
        tokens_batch.append(tokens)
    return tags_batch, tokens_batch


def eval_conll(model, session, tokens, tags, short_report=True):
    """Computes NER quality measures using CONLL shared task script."""

    y_true, y_pred = [], []
    for x_batch, y_batch, lengths in batches_generator(1, tokens, tags):
        tags_batch, tokens_batch = predict_tags(model, session, x_batch, lengths)
        if len(x_batch[0]) != len(tags_batch[0]):
            raise Exception("Incorrect length of prediction for the input, "
                            "expected length: %i, got: %i" % (len(x_batch[0]), len(ta
gs_batch[0])))
        predicted_tags = []
        ground_truth_tags = []
        for gt_tag_idx, pred_tag, token in zip(y_batch[0], tags_batch[0], tokens_batc
h[0]):
            if token != '<PAD>':
                ground_truth_tags.append(idx2tag[gt_tag_idx])
                predicted_tags.append(pred_tag)

        # We extend every prediction and ground truth sequence with 'O' tag
        # to indicate a possible end of entity.
        y_true.extend(ground_truth_tags + ['O'])
        y_pred.extend(predicted_tags + ['O'])

    results = precision_recall_f1(y_true, y_pred, print_results=True, short_report=sh
ort_report)
    return results

"""## Run your experiment
```

```
Create *BiLSTMModel* model with the following parameters:
 - *vocabulary_size* — number of tokens;
 - *n_tags* — number of tags;
 - *embedding_dim* — dimension of embeddings, recommended value: 200;
 - *n_hidden_rnn* — size of hidden layers for RNN, recommended value: 200;
 - *PAD_index* — an index of the padding token (`<PAD>`).

Set hyperparameters. You might want to start with the following recommended values:
- *batch_size*: 32;
- 4 epochs;
- starting value of *learning_rate*: 0.005
- *learning_rate_decay*: a square root of 2;
- *dropout_keep_probability*: try several values: 0.1, 0.5, 0.9.

However, feel free to conduct more experiments to tune hyperparameters and earn extra
 points for the assignment.
"""

tf.compat.v1.reset_default_graph()
tf.compat.v1.disable_eager_execution()

model = BiLSTMModel(vocabulary_size=len(token2idx), n_tags=len(tag2idx), embedding_di
m=200, n_hidden_rnn=200, PAD_index=token2idx['<PAD>'])

batch_size = 32
n_epochs = 4
learning_rate = 0.005
learning_rate_decay = np.sqrt(2)
dropout_keep_probability = 0.5

"""If you got an error *"Tensor conversion requested dtype float64 for Tensor with dt
ype float32"* in this point, check if there are variables without dtype initialised.
Set the value of dtype equals to *tf.float32* for such variables.

Finally, we are ready to run the training!
"""

sess =  tf.compat.v1.Session()
sess.run( tf.compat.v1.global_variables_initializer())

print('Start training... \n')
for epoch in range(n_epochs):
    # For each epoch evaluate the model on train and validation data
    print('-' * 20 + ' Epoch {} '.format(epoch+1) + 'of {} '.format(n_epochs) + '-
' * 20)
    print('Train data evaluation:')
    eval_conll(model, sess, train_tokens, train_tags, short_report=True)
    print('Validation data evaluation:')
    eval_conll(model, sess, validation_tokens, validation_tags, short_report=True)
    # Train the model
    for x_batch, y_batch, lengths in batches_generator(batch_size, train_tokens, trai
n_tags):
        model.train_on_batch(sess, x_batch, y_batch, lengths, learning_rate, dropout_
keep_probability)

    # Decaying the learning rate
    learning_rate = learning_rate / learning_rate_decay

print('...training finished.')

"""Now let us see full quality reports for the final model on train, validation, and
test sets. To give you a hint whether you have implemented everything correctly, you
might expect F-score about 40% on the validation set.

**The output of the cell below (as well as the output of all the other cells) should
be present in the notebook for peer2peer review!**
"""

print('-' * 20 + ' Train set quality: ' + '-' * 20)
train_results = eval_conll(model, sess, train_tokens, train_tags, short_report=False)

print('-' * 20 + ' Validation set quality: ' + '-' * 20)
validation_results = eval_conll(model, sess, validation_tokens, validation_tags, shor
t_report=False)

print('-' * 20 + ' Test set quality: ' + '-' * 20)
test_results = eval_conll(model, sess, test_tokens, test_tags, short_report=False)

"""### Conclusions

Could we say that our model is state of the art and the results are acceptable for th
e task? Definately, we can say so. Nowadays, Bi-
LSTM is one of the state of the art approaches for solving NER problem and it outperf
orms other classical methods. Despite the fact that we used small training corpora (i
n comparison with usual sizes of corpora in Deep Learning), our results are quite goo
d. In addition, in this task there are many possible named entities and for some of t
hem we have only several dozens of trainig examples, which is definately small. Howev
er, the implemented model outperforms classical CRFs for this task. Even better resul
ts could be obtained by some combinations of several types of methods, e.g. see [this
](https://arxiv.org/abs/1603.01354) paper if you are interested.
"""
```

Conclusion

Nowadays, Bi-LSTM is one of the state-of-the-art approaches for solving NER problem and it outperforms other classical methods. Even though we used small training corpora (in comparison with usual sizes of corpora in Deep Learning), our results are quite good. In addition, in this task there are many possible named entities and for some of them we have only several dozens of training examples, which is small. However, the implemented model outperforms classical CRFs for this task.