**Department of Computer Engineering**

**Academic Term: Jan-April 2021**

**Rubrics for Assignments**

**Class : B*.E. Computer* Subject Name: NLP**

**Semester : VIII Subject Code: CSL804**

|  |  |
| --- | --- |
| **Assignment No:** | 1 |
| **Title:** | Industry or Domain-specific Case Study of one Natural Language Processing Application |
| **Date of Performance:** | 23/03/2021 |
| **Roll No:** | 8364 |
| **Name of the Student:** | Vedant Sahai |

**Rubrics for Assignment Grading:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Indicator** | **Very Poor** | **Poor** | **Average** | **Good** | **Excellent** |
| **Timeline (2)** | More than three sessions late (0) | More than two sessions late (0.5) | Two sessions late (1) | One session late (1.5) | Early or on time (2) |
| **Organization (3)** | N/A | Very poor readability and not structured (0.5) | Poor readability and somewhat structured (1) | Readable with one or two mistakes and structured (2) | Very well written and structured without any mistakes (3) |
| **Level of content (3)** | N/A | Major points are omitted or  addressed minimally (0.5) | All major topics are covered, the information is accurate. (1) | Most major and some minor criteria are included. Information is  Accurate (2) | All major and minor criteria are covered and are accurate. (3) |
| **Depth of Knowledge (2)** | N/A | One answer corrects (0.5) | Two answers correct (1) | Three answers correct (1.5) | Four answers correct (2) |

**Total Marks :**

**Signature of the Teacher :**

**Chatbot Development for Healthcare Industry**

**THE CLIENT**

As the global pandemic puts stress on the healthcare system, the need for telehealth chatbots, catering to a growing number of patients, has gone high. Keeping this in mind, the US-based IT company, SEVA Technologies LLC, wanted to develop a telehealth bot that would help inpatient care during these trying times.

**THE CHALLENGE**

SEVA Technologies was looking for an AI-based iOS mobile app that would be a smart digital assistant to patients and healthcare staff at hospitals. To develop this mobile app, they approached Capital Numbers. The US-based company is our existing client, and they have immense faith in our high-quality service. Therefore, when they planned to build this AI bot, they contacted us without any second-guessing. While discussing the project with the client, Capital Numbers found out that they already have a bot in place; however, it was not programmed correctly, plus, the bot was outdated. Our biggest challenge was to enhance this existing bot platform with a cutting-edge AI solution without changing the existing codebase. Besides, with the rising COVID-19 cases and piling pressure on healthcare systems, we had to come up with an evolved telehealth bot within a very short time. Our challenge was to conduct in-depth research within a few weeks to develop a bot that transforms patient diagnoses.

**TECHNOLOGY STACK**

* Google Dialogflow
* Swift
* REST API
* MongoDB

**THE SOLUTION**

* UI/UX
* Backend Dev
* Front-end Dev
* SEO

We held multiple meetings with our client to dive into the project needs in detail. Through a strategic approach, our qualified tech team worked towards developing an AI-based iOS app that can identify patients’ needs and pop-up alerts to prompt caregivers to take corrective actions on time. We chose Google Dialogflow to develop this conversational bot. Dialogflow is a platform that makes it easy to integrate conversational UI into a mobile app. It analyzes text and audio inputs from users. Accordingly, it responds to users, either through speeches or texts. As a result, Dialogflow was our obvious choice for this interactive voice response system. Next, our experts used Swift because of its clean code and simplified syntax. Moreover, Swift comes with dynamic libraries that get updated independently, speed up load time, and enhance overall app functionality. Also, Swift offers code reusability, removes classes of unsafe code, makes apps secure, and comes with UTF-8 based encoding that supports multiple languages and emojis. Therefore, this programming language was a top choice for the iOS platform. We used REST API because this architectural style comes with various layers that assure security and flexibility to iOS apps. REST can also easily integrate with existing platform architecture without the need to refactor the codebase from scratch. Plus, REST APIs are stateless that makes iOS apps scalable and cacheable. To add versatility to the app, we chose the schema-less data structure - MongoDB. This database comes with auto-sharding that helps in horizontal scaling. Moreover, MongoDB adds flexibility and high speed to apps. It is developer-friendly and one of the most cost-effective options. MongoDB was the perfect fit for our iOS app.

**RESULTS**

By implementing the latest technologies, our technical specialists came up with an innovative healthcare solution that always keeps patients and caregivers connected. We successfully created the iOS platform, called the SeVa app, that signals hospital staff if patients require medical or clinical assistance.

* User Login

Patients need to download the SeVa app on their iPads, log in with username and password, and interact with the nurses within the chat interface.

* Unique Patient ID

When a patient fills in his login credentials in the SeVa app, a unique patient ID gets generated along with his room number at the hospital.

* Personalized Care

Once the unique ID gets generated, patients can start accessing the app by keeping their iPads beside their hospital beds to call nurses for various kinds of assistance such as anxiety, stress, drugs, or other personal care

* Timely Alerts and Signals

The emotionally intelligent bot keeps track of a patient's condition by efficiently asking the patient about his health at regular intervals. When a patient specifies some sickness or discomfort, the voice assistant in the iOS app immediately sends real-time alerts to the caregivers. The nurses or caregivers receive this alert on their iPads and extend help immediately without delay.

* Seamless Bot Flow

Our experts used the best conversational datasets to train the bot with relevant questions to improve the quality of patient care. Through Natural Language Processing, we triggered a blot flow that sounds human-like while asking patients questions like:

1. Are you experiencing any pain?
2. Do you have the urge to use the restroom at this time?
3. Does your brace need to be readjusted?
4. Would you like a heat pack or an ice pack while your nurse gets your medicine?

* Innovation in Healthcare

By deploying this virtual assistant app on time, Capital Numbers not only triggered a conversational bot flow but also triggered innovation in healthcare. This instilled greater client trust. We couldn’t be prouder of our skilled developers who created this patient-care app that offers trusted healthcare services and enables efficient crisis management in these uncertain times.

* Cutting-edge Tech Stack

Our dedicated team members were quick, prompt, and proficient at optimizing the bot performance with built-in technologies like Swift 5 and MongoDB that leave ample room for future scalability. Moreover, our developers' readiness to research on Google Dialogflow and implement it to enable natural language conversations was highly commendable

* Short Timeline

The ability to build an AI-powered virtual assistant within two months and port the same to Google based iOS app solution was something we excelled at.

* Client Testimonial

Our pool of talents delivered a bug-free iOS version of the AI bot that assists patients with medical guidance, symptom checks, medical queries, nutrition, and other tailored services. The client acknowledged the efficacy with which Capital Numbers could deliver this Machine Learning project that ensures 100% accuracy. They added: “The pool of resources is amazing. They can find you any skill set if needed.”

**Department of Computer Engineering**

**Academic Term: Jan-April 2021**

**Rubrics for Assignments**

**Class : B*.E. Computer* Subject Name: NLP**

**Semester : VIII Subject Code: CSL804**

|  |  |
| --- | --- |
| **Assignment No:** | 2 |
| **Title:** | Discuss reference resolution problem with suitable example. Explain Lappin and Leass’s algorithm for pronoun resolution with a suitable example. |
| **Date of Performance:** | 26/04/2021 |
| **Roll No:** | 8364 |
| **Name of the Student:** | Vedant Sahai |

**Rubrics for Assignment Grading:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Indicator** | **Very Poor** | **Poor** | **Average** | **Good** | **Excellent** |
| **Timeline (2)** | More than three sessions late (0) | More than two sessions late (0.5) | Two sessions late (1) | One session late (1.5) | Early or on time (2) |
| **Organization (3)** | N/A | Very poor readability and not structured (0.5) | Poor readability and somewhat structured (1) | Readable with one or two mistakes and structured (2) | Very well written and structured without any mistakes (3) |
| **Level of content (3)** | N/A | Major points are omitted or  addressed minimally (0.5) | All major topics are covered, the information is accurate. (1) | Most major and some minor criteria are included. Information is  Accurate (2) | All major and minor criteria are covered and are accurate. (3) |
| **Depth of Knowledge (2)** | N/A | One answer corrects (0.5) | Two answers correct (1) | Three answers correct (1.5) | Four answers correct (2) |

**Total Marks :** **Signature of the Teacher :**

**Q. Discuss reference resolution problems with suitable examples. Explain Lappin and Leiss’s algorithm for pronoun resolution with a suitable example.**

First, let us understand what reference expressions are.

* A natural language expression used to perform reference is called a referring expression and the entity that is referred to is called the referent.
* E.g. - John went to Bill’s car dealership to check out an Acura Integra. He looked at it for about an hour.
* Two referring expressions that are used to refer to the same entity are said to co-refer (John and he)
* Reference to an entity that has been previously introduced into the discourse is called anaphora and the referring expression used is said to be anaphoric. (he and it)
* Types of reference expressions:

1. *Indefinite Noun-Phrases* - These introduce entities that are new to the hearer into the discourse context. For eg: a, an
2. *Definite Noun-Phrases* - These are used to refer to an entity that is identifiable to the listener because it has been mentioned in the discourse context
3. *Pronouns* - This is another form of definite reference and refers to entities that were introduced one or two sentences back in the ongoing discourse
4. *Demonstrative Pronouns* - These pronouns are associated with spatial proximity indicating closeness and signalling distance. For e.g.: this, that.

Reference Resolution:

*Jack and jill when up and Jack fell and broke his crown*

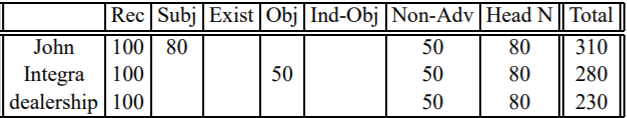
Here the word ‘his’ refers to jack, but the computer cannot understand this because it lacks common sense.

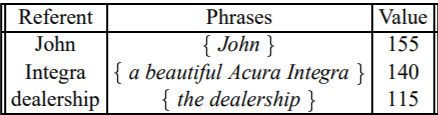
* When 2 or more expressions in the text refer to one person or thing, then they are said to co-refer and this task of finding all expressions that are co-referenced with any of the entities found in a text is known as coreference resolution hence to derive the correct interpretation of the text pronouns and other expressions must be resolved.

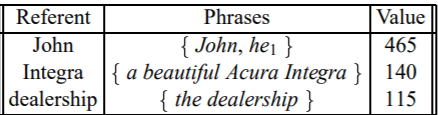
Lappin and Leass Algorithm:

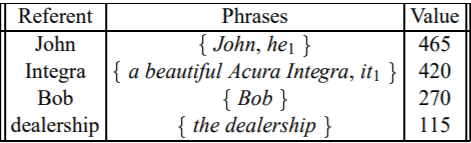
* Lappin and Leass (1994) describe a straightforward algorithm for pronoun interpretation that takes many of these into consideration.
* The algorithm employs a simple weighting scheme that integrates the effects of the recency and syntactically-based preferences; no semantic preferences are employed beyond those enforced by agreement.
* Two types of operations performed by the algorithm: discourse model update and pronoun resolution
* Discourse model update: Noun phrase that evokes a new entity is encountered, a representation for it must be added to the discourse model, and a degree of salience computed for it
* Pronoun resolution: The weights that each factor assigns to an entity in the discourse model are cut in half each time a new sentence is processed. This, along with the added effect of the sentence recency weight captures the Recency preference
* Example:

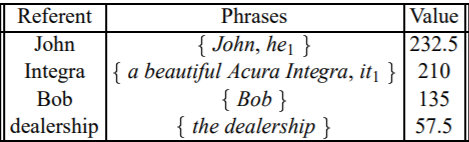
SENTENCE: John saw a beautiful Acura Integra at the dealership. He showed it to Bob. He bought it.

Collect potential referents and compute their initial salience values. Fig.1. shows the contribution to the salience of each of the salience factors.

There are no pronouns to be resolved in this sentence, so we move on to the next, degrading the above values by a factor of two.

 First, the pronoun he is added in the equivalence class for John. Since he occurs in the current sentence and John in the previous one, the salience factors do not overlap between the two.

The next noun phrase in the second sentence is the pronoun it, which is compatible with the Integra or the dealership. We first need to compute the final salience values by adding the applicable weight

We again degrade the current weights by one-half. The weights used by Lappin and Leass were arrived at by experimentation on a development corpus of computer training manuals.

**Department of Computer Engineering**

**Academic Term: Jan-April 2021**

**Rubrics for Assignments**

**Class : B*.E. Computer* Subject Name: NLP**

**Semester : VIII Subject Code: CSL804**

|  |  |
| --- | --- |
| **Assignment No:** | 3 |
| **Title:** | Predict tags on StackOverflow with linear models or Recognize named entities on Twitter with LSTMs |
| **Date of Performance:** | 30/04/2021 |
| **Roll No:** | 8364 |
| **Name of the Student:** | Vedant Sahai |

**Rubrics for Assignment Grading:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Indicator** | **Very Poor** | **Poor** | **Average** | **Good** | **Excellent** |
| **Timeline (2)** | More than three sessions late (0) | More than two sessions late (0.5) | Two sessions late (1) | One session late (1.5) | Early or on time (2) |
| **Organization (3)** | N/A | Very poor readability and not structured (0.5) | Poor readability and somewhat structured (1) | Readable with one or two mistakes and structured (2) | Very well written and structured without any mistakes (3) |
| **Level of content (3)** | N/A | Major points are omitted or  addressed minimally (0.5) | All major topics are covered, the information is accurate. (1) | Most major and some minor criteria are included. Information is  Accurate (2) | All major and minor criteria are covered and are accurate. (3) |
| **Depth of Knowledge (2)** | N/A | One answer corrects (0.5) | Two answers correct (1) | Three answers correct (1.5) | Four answers correct (2) |

**Total Marks :**

**Signature of the Teacher :**

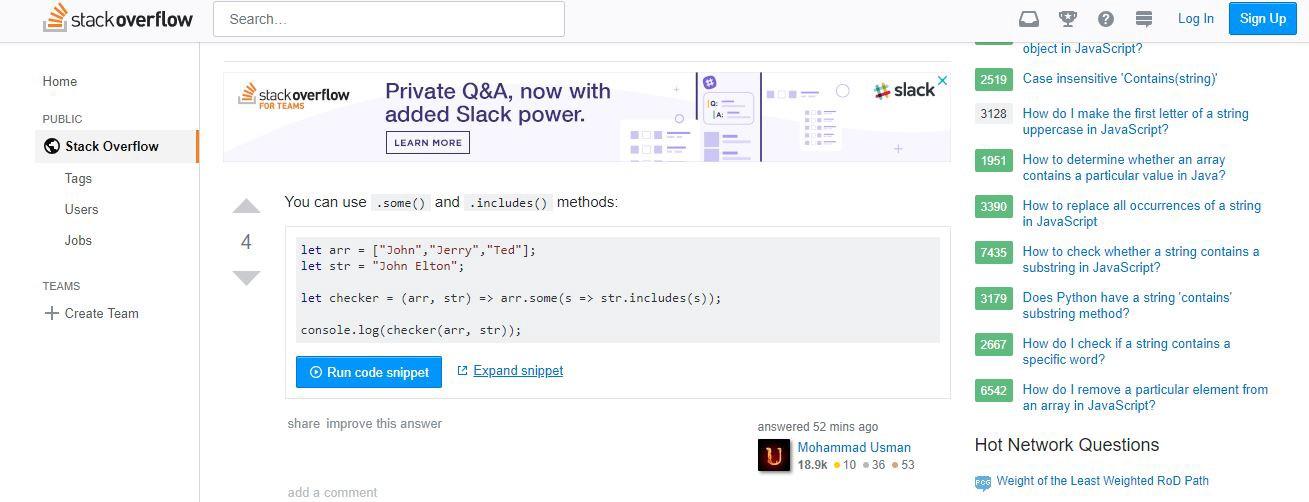
**Assignment 3.1**

**Problem Statement:**  Predict tags on Stack Overflow with linear models

**Link**: <https://drive.google.com/drive/folders/1jPDPq35eVfUVQdIL7Sy8chHTJfbcVq3s?usp=sharing>

**Theory:**

One of the most common tasks of NLP is to automatically predict the topic of a question. In this assignment, we’ll start by pre-processing 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.

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

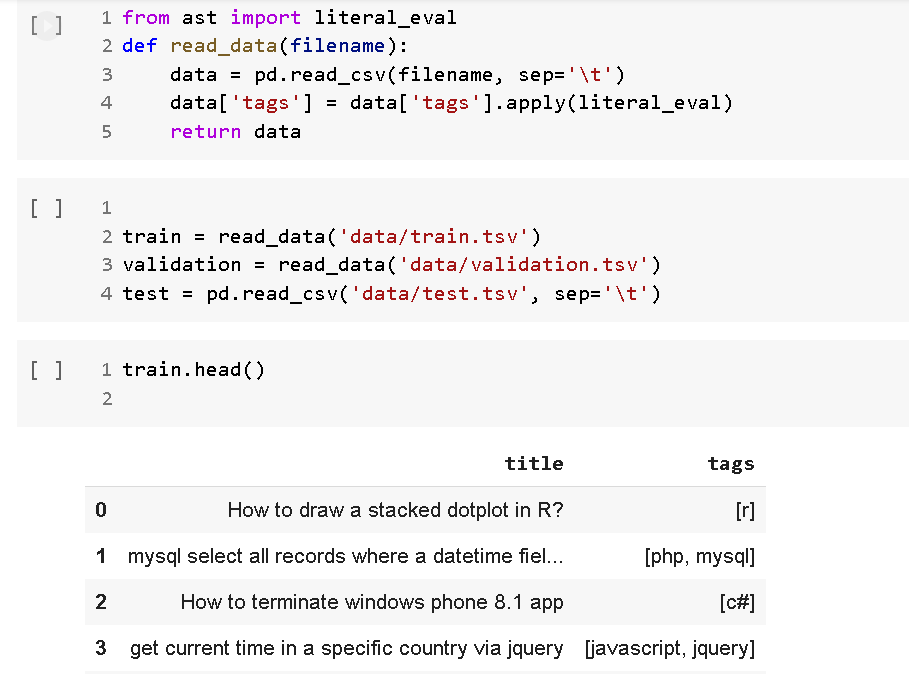
Stack overflow detects that the user who has answered the question has already done so to similar questions related to JavaScript, Strings, Arrays etc in the past and recognizes that he is an expert in the subject.  People can provide the tags related to the question on their own or Stack Overflow can predict the tags using the text in the 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

* Perform Clean-up

*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

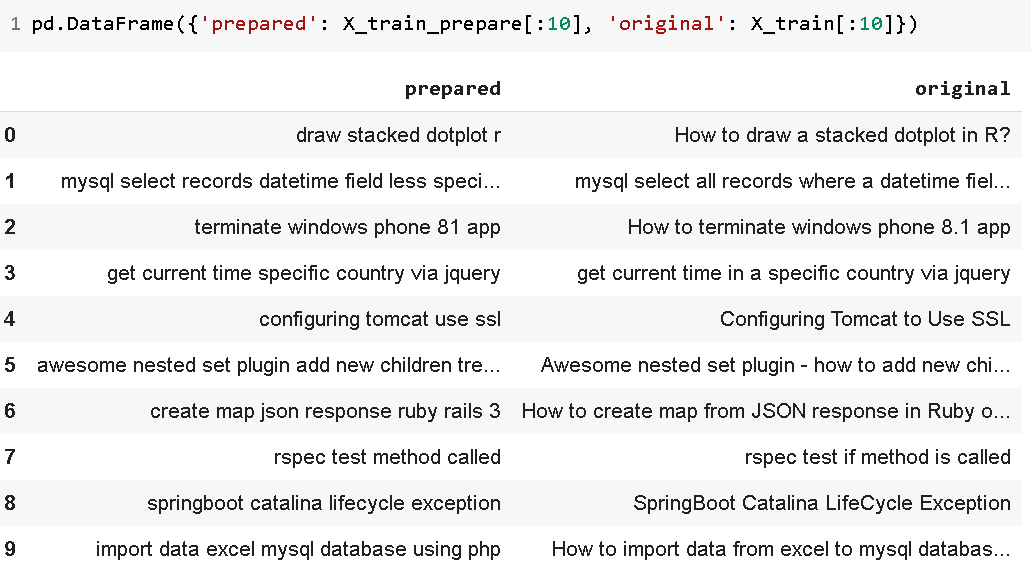
*Step 2: STOPWORD REMOVAL*

Stop words refer to the most common word which while providing 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 to increase the size of the dataset instead of removing the words.

* 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 its Term Frequency

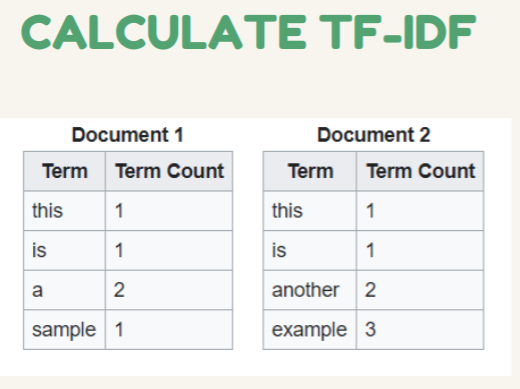
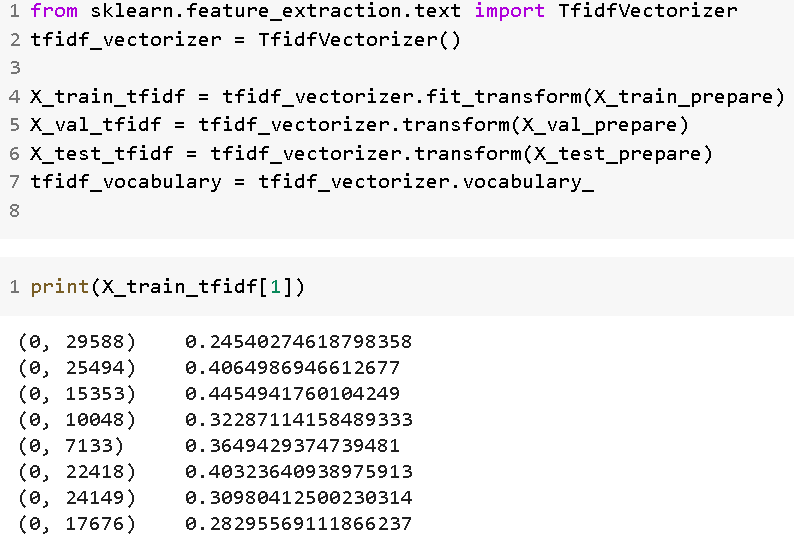
*INVERSE DOCUMENT FREQUENCY*

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).

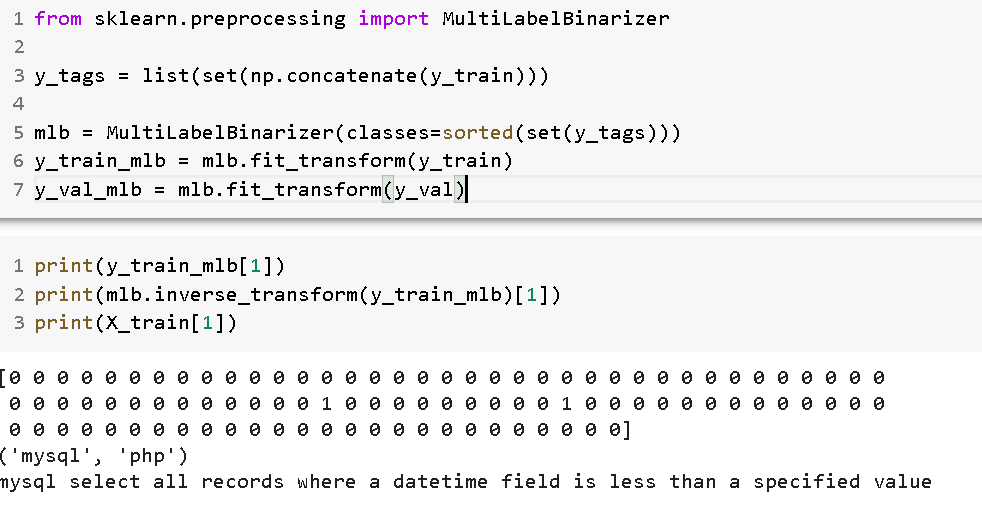
*TF-IDF*

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

Multiply TF \* IDF



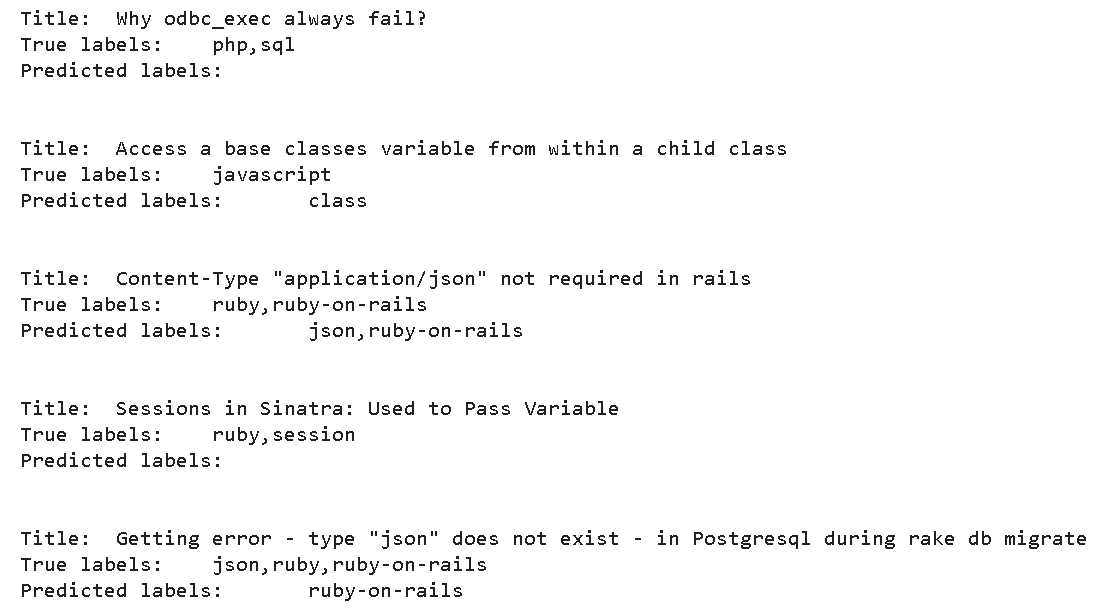
* Convert to Multi-label Binarizer

Helps make it easier for Linear Model to comprehend the topic

* 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

**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 Collaboratory.

The 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\_english])

    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') \* 100

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 pre-processing and linear algorithms, we were successfully able to distinguish tags on stack overflow questions.

**Assignment 3.2**

# **Problem Statement:** Recognize named entities on Twitter with LSTMs

**Link**: <https://drive.google.com/file/d/119E7Y6OJGRaSf2dTsMmKrCtyj2HHn4QU/view?usp=sharing>

**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:

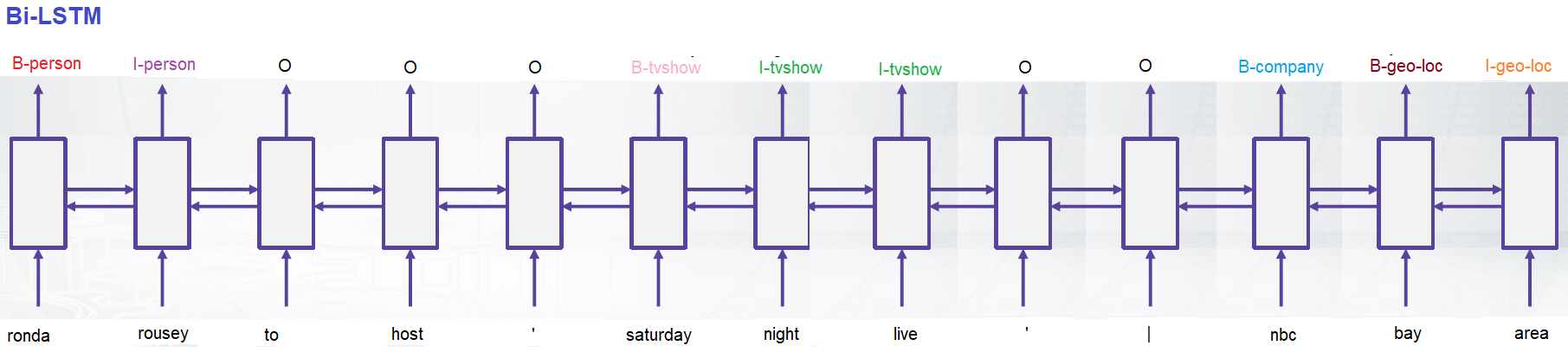
B-PER I-PER    O     O   B-ORG I-ORG

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 consequent entities with similar types. More examples are shown below in the diagram

A solution to 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:**

# -\*- coding: utf-8 -\*-

"""assignment3-Twitter-NER.ipynb

Automatically generated by Collaboratory.

The 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 the Named Entity Recognition (NER) problem. 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 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 to 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 tutorials during our work on this assignment, e.g. [this one] (https://www.tensorflow.org/tutorials/recurrent) could be a good starting point.

### Data

The following cell will download all data required for this assignment into the folder `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 whitespace. Different tweets are separated by an empty line.

The function \*read\_data\* reads a corpus from the \*file\_path\* and returns two lists: one 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 the `<USR>` token and any URL to the `<URL>` token. We could think that a URL and a nickname are just strings that start 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 can 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 output 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 sentences.

"""

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 ids 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 lengths, 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 architecture based on TensorFlow building blocks. It's as fun and easy as a lego constructor! We will create an LSTM network that will produce probability distribution over tags for each token in a sentence. To take into account both the right and left contexts of the token, we will use Bi-Directional LSTM (Bi-LSTM). A 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 change 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], name='input\_batch')

    self.ground\_truth\_tags =  tf.compat.v1.placeholder(dtype=tf.int32, shape=[None, None], name='ground\_truth\_tags')

    # Placeholder for lengths of the sequences.

    self.lengths =  tf.compat.v1.placeholder(dtype=tf.int32, shape=[None], name='lengths')

    # 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 some preparatory steps:

- Create embeddings matrix with [tf.Variable](https://www.tensorflow.org/api\_docs/python/tf/Variable). Specify its name (\*embeddings\_matrix\*), type  (\*tf.float32\*), and initialize with random values.

- Create forward and backward LSTM cells. TensorFlow provides several RNN cells ready for you. We suggest that you use \*LSTMCell\*, but you can also experiment with other 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 neural networks. Specify all keep probabilities using the dropout placeholder that we created 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) embeddings 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 backward 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 the loss function.

In case you need to debug something, the easiest way is to check that tensor shapes of 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.sqrt(embedding\_dim)

    embedding\_matrix\_variable = tf.Variable(initial\_embedding\_matrix, dtype=tf.float32)

    # 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\_cell.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(cell\_fw=forward\_cell,

                                                                        cell\_bw=backward\_cell,

                                                                        inputs=embeddings,

                                                                        sequence\_length=self.lengths,

                                                                        dtype=tf.float32)

    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 most 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 function. 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=ground\_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/AdamOptimizer) optimizer with a learning rate from the corresponding placeholder.

You will also need to apply clipping to eliminate exploding gradients. It can be easily done with [clip\_by\_norm](https://www.tensorflow.org/api\_docs/python/tf/clip\_by\_norm) 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\_rate\_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 self.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 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 that 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 actual data through the placeholders that we defined before.

"""

def train\_on\_batch(self, session, x\_batch, y\_batch, lengths, learning\_rate, dropout\_keep\_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 start 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 and 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(tags\_batch[0])))

        predicted\_tags = []

        ground\_truth\_tags = []

        for gt\_tag\_idx, pred\_tag, token in zip(y\_batch[0], tags\_batch[0], tokens\_batch[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=short\_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\_dim=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 dtype 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, train\_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 a train, validation, and test sets. To give you a hint whether you have implemented everything correctly, you might expect an F-score of 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, short\_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 the task?  We can say so. Nowadays, Bi-LSTM is one of the states of the art approaches for solving NER problem and it outperforms other classical methods. Even though we used small training corpora (in comparison with the 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. Even better results 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 the 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.